Proceedings Paper:
Srivisut, Komsan, Clark, John A. orcid.org/0000-0002-9230-9739 and Paige, Richard F. orcid.org/0000-0002-1978-9852 (2018) Dependent input sampling strategies:using metaheuristics for generating parameterised random sampling regimes. In: GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference. 2018 Genetic and Evolutionary Computation Conference, GECCO 2018, 15-19 Jul 2018 ACM , JPN , pp. 1451-1458.

https://doi.org/10.1145/3205455.3205495

Dependent Input Sampling Strategies

Using Metaheuristics for Generating Parameterised Random Sampling Regimes

Komsan Srivisut
University of York
Department of Computer Science
York, UK
ks1077@york.ac.uk

John A. Clark
University of Sheffield
Department of Computer Science
Sheffield, UK
john.clark@sheffield.ac.uk

Richard F. Paige
University of York
Department of Computer Science
York, UK
richard.paige@york.ac.uk

ABSTRACT

Understanding extreme execution times is of great importance in gaining assurance in real-time embedded systems. The standard benchmark for dynamic testing—uniform randomised testing—is inadequate for reaching extreme execution times in these systems. Metaheuristics have been shown to be an effective means of directly searching for inputs with such behaviours but the increasing complexity of modern systems is now posing challenges to the effectiveness of this approach. The research reported in this paper investigates the use of metaheuristic search to discover *biased random sampling regimes*. Rather than search for test inputs, we search for *distributions* of test inputs that are then sampled. The search proceeds to discover and exploit relationships between test input variables, leading to sampling regimes where the distribution of a sampled parameter depends on the values of previously sampled input parameters. Our results show that test vectors indirectly generated from our *dependent* approach produce significantly more extreme (longer) execution times than those generated by direct metaheuristic searches.

CCS CONCEPTS

• **Theory of computation → Optimization with randomized search heuristics**; • **Software and its engineering → Software testing and debugging**; **Search-based software engineering**; • **Computer systems organization →** *Multicore architectures*;

KEYWORDS

genetic algorithms, hill climbing, metaheuristics, simulated annealing, temporal testing

ACM Reference Format:
Komsan Srivisut, John A. Clark, and Richard F. Paige. 2018. Dependent Input Sampling Strategies: Using Metaheuristics for Generating Parameterised Random Sampling Regimes. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205495

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205495

1 INTRODUCTION

Metaheuristic algorithms have been established as one of the most practical and effective approaches to optimisation problems [2]. In search-based software testing their most common application is automatic test data generation, i.e. searching for test inputs whose execution satisfies some property of interest [14]. The test data is usually the *direct* target of the searches—the metaheuristic's search space is the space of test inputs.

Randomisation also plays an important role in test data generation. The most common form of random test data generation is to sample the test data *uniformly* and *independently* from the domains of the input parameters. If these input domains are $D_1, D_2, \ldots, D_n$, then each domain D_i is sampled in turn to produce a test vector $(t_1, t_2, \ldots, t_n)$. When sampling the domain D_i, all possible elements of that domain have the same chance of being selected; this is what is meant by *uniform* sampling. Moreover, the sampling of one domain D_i is not affected by sampling of any different domain D_j; this is what is meant by *independent* sampling. This is the simplest approach and many languages include library functionality that can be used for such sampling, e.g. the Java `Random` class's `nextInt()` method for sampling integer values.

However, uniform independent sampling is unlikely to be the most effective approach to discovering optimal test data, particularly where non-functional properties are concerned. For instance, in *temporal testing*, where the aim is to find test inputs that will cause the system to violate timing performance requirements [5], extreme execution times may be concentrated in small partitions of the input space and much of the input space may give unexceptional execution times. To increase the chances of inputs with extreme times being sampled, it would be beneficial to constrain in some way the sampled inputs to more productive regions. However, we generally cannot identify such regions confidently.

Suppose a program with two parameters incurs extreme execution times when both parameters are large or when both are small. A *good* choice for the second parameter depends on what value was sampled for the first and so *dependent* sampling may offer a more efficient means of gaining appropriate test data in this case.

Accordingly, this paper presents an approach that allows restriction to a subset of the input domain and which also allows the sampling distribution for a parameter to depend on the sampled values of earlier ones. The idea of the proposed approach is to shift the target of optimisation from test inputs to strategies for test input generation. Particularly, we target the generation of parameterised random sampling regimes. Search is used to optimise the parameters of a class of such regimes, which can then be used to generate test data. Hence, our metaheuristic operates on the search space

of parameterised distributions. This can be viewed as an *indirect* approach to test data generation.

The contributions in this paper are:

- Demonstration of how optimisation can be used to find an approach for generating a test input from a subset of the input domain and where the sampling distribution for a parameter is dependent on the sampled values of antecedent parameters. We term this a *dependent input sampling strategy*.
- Provision of empirical evidence to show that using the dependent input sampling strategies discovered is more effective than a uniformly independent sampling approach for sampling test inputs that will give rise to *extreme execution times* for numerical functions running on an embedded multicore system.

The remainder of this paper is organised as follows. Section 2 outlines temporal testing, our target application domain. A description of the proposed approach is provided in Section 3. The empirical methodology is presented in Section 4 and the experimental results are subsequently reported and discussed in Section 5. Section 6 concludes the paper and suggests future work.

2 TEMPORAL TESTING

In many real-time embedded systems, especially safety-critical systems, the correctness of their functions depends not only on logical correctness but also on temporal correctness [5, 16] as violations of timing constraints—either outputs produced too early or too late—may be fatal to human life [3]. In order to verify the temporal behaviour of the systems, timing analysis is generally performed by determining the worst-case execution time (WCET) of a computational task to show that such the stringent timing constraints are satisfied [7, 20]. (Strictly, WCET values and a *schedulability model* determine whether response times are satisfied [3].)

One of the main approaches for estimating WCETs is the dynamic (or measurement-based) approach, where a task is run with a set of inputs on a real hardware or processor simulator and the resulting information is used to provide an *estimate* of the WCET [20]. Such approaches are also known as *temporal testing* [5, 14], whose goal is to find test inputs that will cause the system to violate performance timing requirements [5]. Search-based optimisation algorithms, such as genetic algorithms (GAs) and simulated annealing (SA), have been shown to be an effective means of identifying temporal failures in embedded or complex systems [1, 5]. Note that the actual WCET for a task must be at least as long as any witnessed or measured execution time.

Uniform independent random testing is a widely used benchmark for evaluating these approaches. Notwithstanding the achievements of those previous search-based temporal testing approaches, as systematically reviewed in [1, 8], uniform independent random sampling is rarely a good way of discovering test data with extremal properties on embedded or related complex systems: a better benchmark approach is required for comparison and below we describe our non-uniform and dependency based sampling approach.

3 DEPENDENT INPUT SAMPLING STRATEGIES

Sampling uniformly across a full domain is unlikely to be an effective way to reveal optimal test inputs in general, and temporal test vectors in particular. Two clear means of improving efficiency suggest themselves: allowing subdomains to be sampled rather the full domain and allowing such subdomains to be sampled non-uniformly. There is often an assumption that the sampled domain is in some sense contiguous, e.g. integer values sampled from a single full domain of all 2^{32} possible `int` values by Java's `nextInt()` method. However, this is not essential and it is simple to have a sampling domain that is the union of two or more subdomains.

Consistent with the above, the sampling subdomain for any parameter will be a union of a number of domain subranges (intervals), e.g. if the full input domain is $[-100, 100]$, we might have $[-32, 17] \cup [54, 98]$ as the subdomain actually sampled. We do not know in advance which constituent intervals work best but will seek to discover this as part of the search. Also, the constituent intervals can overlap — we allow the search process to discover whether disjoint or overlapping intervals are best. As indicated below, to sample from a subdomain we first select one of its intervals (according to some probability distribution) and then sample uniformly from that interval. Thus, the probability of a value being selected depends on the probabilities of intervals containing that value being selected and the sizes of those intervals. A value that is in the intersection of two intervals has greater likelihood of being selected than a value that is in a single interval alone. For example, it follows that $[0, 2] \cup [0, 2]$ gives rise to a different sampling distribution than $[0, 1] \cup [1, 2]$, even though the overall sampled domain is $[0, 2]$. In the first, the three values $0, 1, 2$ are sampled with equal probability and in the second 1 has the greatest chance of selection (assuming interval selection probabilities are non-zero).

Our approach will also allow the sampling distribution of an input subdomain to depend on the values of subdomains sampled earlier. Regarding the example of input domains for the test vector $t_1, t_2, \ldots, t_n$ as previously described in Section 1, for example, the sampling distribution of D_3 will depend on the sampled values of D_1 and D_2. More precisely, the sampling of a parameter will depend on the *specific intervals* sampled for previous parameters, rather than the exact values sampled. The overall sampling regime can be seen to be a tree. For instance, Figure 1 shows the tree representation of a sampling distribution for a three parameter (A, B, C) problem. For simplicity, we assume here that the sampling domain for each parameter is the union of two intervals.

In particular, in Figure 1, we assume that the first parameter sampled (A) comprises two intervals $[L_{A_0}, U_{A_0}]$ and $[L_{A_1}, U_{A_1}]$. The weights W_{A_0} and W_{A_1} represent the relative chances of each interval of A being selected. We will normalise these weights to get the specific probabilities of choosing each of these intervals by using a histogram-based selection method—the proposed algorithm listed in Algorithm 1. Besides, we can see that similar sampling regimes are available for B, but that separate regimes are in place depending on whether the left or right branch was chosen for A. Suppose that intervals A_0, B_{01} and C_{010} are selected, respectively (as highlighted in blue), the test vector of the three parameter (A, B, C) problem will, therefore, be sampled from such selected intervals.

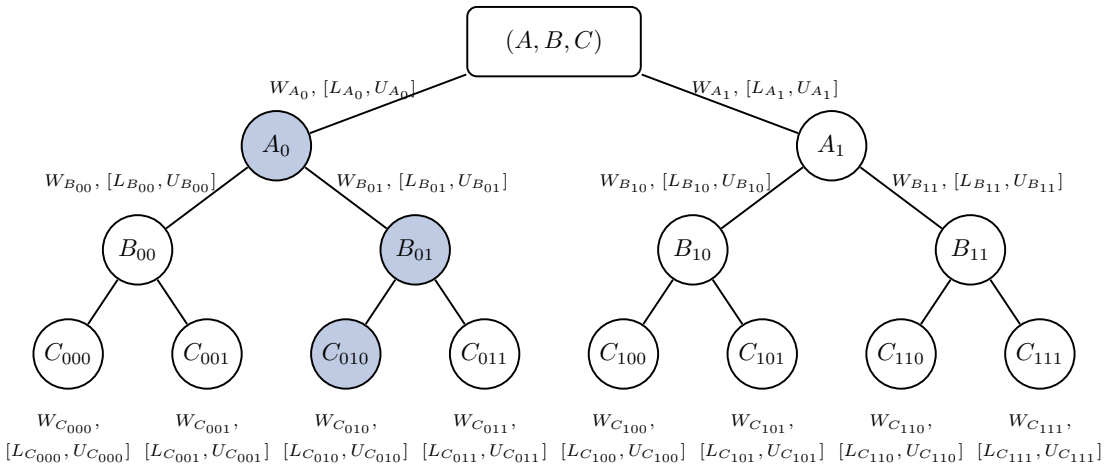


Figure 1: An example of tree structure for correlated distributions

Algorithm 1 A histogram-based selection method

1: $n \leftarrow$ a given total number of intervals of each parameter
2: $W = \{w_0, w_1, w_2, \ldots, w_{n-1}\}$, a set of weights. If $w_i = 0$ for all i then set $w_i = 1$ for all i. (This imposes equal probabilities in this degenerate case and avoids division by 0 under normalisation below.)
3: $total = \sum_{i=0}^{n-1} w_i$, a summation of weights
4: $normalised \leftarrow \emptyset$, an initial set of normalised weights
5: $cumulative \leftarrow \emptyset$, an initial set of cumulative weights
6: **for** $i = 0$ **to** $n - 1$ **do**
7: $\quad normalised_i = \frac{w_i}{total}$
8: $\quad$ **if** $i = 0$ **then**
9: $\quad\quad cumulative_i = normalised_i$
10: $\quad$ **else**
11: $\quad\quad cumulative_i = normalised_i + cumulative_{i-1}$
12: $\quad$ **end if**
13: **end for**
14: $index \leftarrow 0$, an initial index
15: $r \leftarrow U(0, 1)$, a random number sampled from the uniform distribution
16: **while** $r > cumulative_{index}$ **do**
17: $\quad index = index + 1$
18: **end while**
19: **return** $index$

Regarding the tree representation described above, let I be the set of input arguments (or a test vector) for a function to be tested. Then $|I|$ is the cardinality of I, i.e. is the number of input arguments. Let c be a given number of interval choices per input argument.

The number of possible paths, denoted P, through the tree is:

$$P = c^{|I|} \tag{1}$$

and the total number of intervals that can be sampled on these paths, denoted by N, is given by:

$$N = \sum_{i=1}^{|I|} c^i \tag{2}$$

In Figure 1, for example, $I = \{A, B, C\}$, $|I| = 3$ and $c = 2$; so that $P = 8$ and $N = 14$.

Rather than use an optimisation technique to directly generate test inputs, our approach *indirectly* generates test inputs by applying a metaheuristic search to construct the tree representation of the sampling distributions and then samples the test inputs from the constructed trees. The search space is the space of such parameterised distributions. (The parameters are those elements read from the genome.)

Our approach can be thought of as a specific type of estimation of distribution algorithm (EDA), where the sampling distribution is the 'model' for the test data. The discovery of effective subdomains has been seen in work by Patrick et al. [15] but not for the target application domain (temporal testing in particular and non-functional property testing in general).

In this paper, the approach is separated into three different strategies: basic, fixed delta-based and randomised delta-based methods.

3.1 The Basic Method

Here each interval is associated with a lower bound L, an upper bound U, and a selection weight w. For each interval these three elements are decoded from the solution genome. Thus, the genome solution length for the basic method is therefore $3N$.

Figure 2 illustrates a genome size of 42 for the three parameter (A, B, C) problem with two interval choices per input argument from Figure 1. The genome is decoded into the tree data structure by using a combination of two tree traversals, i.e. level-order and pre-order, respectively. Our decoding is one of many; an optimal representation is the subject of future research.

3.2 Fixed Delta-Based Method

In the basic method, the three basic parameters are evolved. In the fixed delta-based method we examine the impact of restricting all subdomains' intervals to have the same width. The size of genome for the fixed delta-based approach is $2N$, since each node only requires a lower bound and weight to be assigned; its upper bound

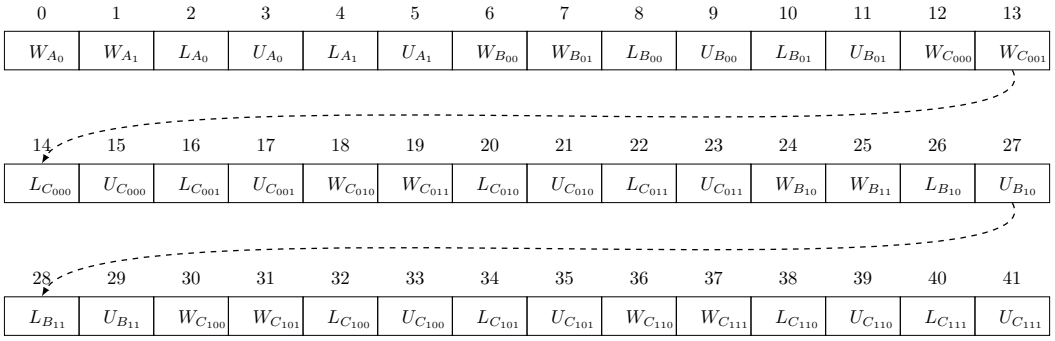


Figure 2: An example of a genome for a basic correlated input approach

is determined by adding a fixed delta value (Δ) to the lower bound. In the three parameter (A, B, C) problem, for example, the genome size for the fixed delta-based method is 28 as depicted in Figure 3.

3.3 Randomised Delta-Based Method

The randomised delta-based approach is more flexible. The aim is to allow the search to be a bit more explorational. A randomised delta value (Δ_r) for each subdomain is individually sampled from a given range, e.g. $[1, 20]$, and is then added to the lower bound to give the subdomain's upper bound. The genome size of any metaheuristic for the randomised delta-based approach is also the same as the fixed delta-based method, i.e. $2N$.

4 EXPERIMENTS

Our experiments used direct and indirect approaches to search for temporal test vectors of a task running on an embedded multicore platform. There are a wide variety of applications of the QorIQ P4080[1] multicore processor (P4080) [17] in industrial real-time embedded systems [11], such as telecommunications and networking [18], and also on safety-critical systems, such as aerospace and defence markets [9]. The P4080 was used as the primary hardware platform in this study. The details of the experiments are given below.

4.1 Software Under Test

In most applications, numerical functions are used to perform basic numeric calculations and are also used as elements of more complex mathematical computations. A number of commercial and academic institutions have provided such numerical functions in the form of scientific libraries. In this paper, a polynomial root-finding routine of the GNU Scientific Library (GSL) [6], which is free software provided by the GNU operating system, is used as a benchmark since it allows us to explore the temporal behaviour with different numbers of input arguments. The function is used to find roots of general polynomial equation in the form of $a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1} = 0$, where the coefficients of the highest order term must be non-zero.

[1]P4080 processor includes eight e500 PowerPC cores scaling to 1.5 GHz, and has a three-level cache-hierarchy: 32KB I/D L1, 128KB private L2 per core and 2 MB shared L3 [17].

Even though all GSL functions are implemented in a single-threaded procedure, they can be used in multi-threaded programs as they are thread-safe [6]. Furthermore, using a single-threaded task on a multicore is actually considered a feasible proposition for some critical environments, where the task is specific to a particular core. Although it clearly does away with many of the advantages of multicore, it also reduces complexity, which is a major criterion in gaining assurance in safety-critical systems.

4.2 Preparation

The experiments are separated into direct and indirect approaches. Both approaches employ a number of metaheuristic algorithms to directly and indirectly generate test vectors for the GSL's polynomial root finder. Metaheuristic techniques include stochastic hill climbing (HC), SA and GA. The experiments were facilitated by the Java-based Evolutionary Computation Research System (ECJ) [13], which is one of the most popular evolutionary computation tool kits, is extensible, and has a clear descriptive manual and strong community support [19]. ECJ version 23 was used since it was the latest release available at the time the experiments were conducted. As a result, the $(\mu + \lambda)$–evolution strategy (ES) feature was adapted and modified for the experiments of single-state approaches.

Particularly, a simple HC and SA could be considered as the degenerate cases, i.e. (1 + 1)–ES [12]. A common mutator for a vector individual was used to generate a candidate solution for these two single-stage approaches with the mutation probability (P_m) of 0.1; this makes the new solution *randomly slightly different* from the current one. Additionally, the geometric reduction cooling function [4, 10], $T_{k+1} = \alpha T_k$, was used for SA. In this paper, the initial temperature T_0 was at 1,000 and cooling rate was 0.99.

A test case of temporal testing was given as a sequence of values (or an integer vector) between −32,768 and 32,767, which equals to a 2-byte signed integer data's range in the C programming language. The rest of the parameter settings for the metaheuristic search algorithms are listed in Table 1. Note that we controlled the time spent running each algorithm by specifying the parameters of such algorithms so that they would take the same number of evaluations.

Direct and indirect approaches use the same metaheuristics parameter settings as mentioned above, excepting for genome sizes,

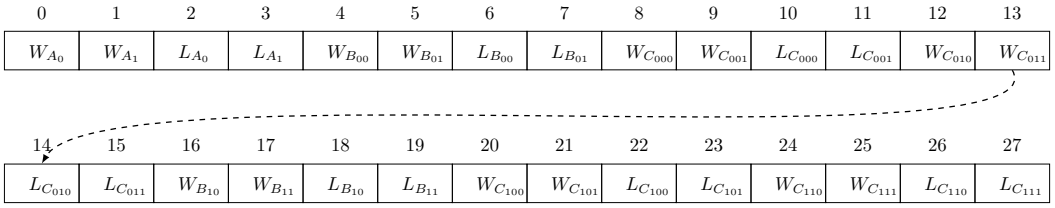


Figure 3: An example of a genome for delta-based correlated input approaches

Table 1: Parameter settings for metaheuristic algorithms

	Algorithm		
Parameter	**HC**	**SA**	**GA**
Generations	10,099	10,099	101
Population size	2	2	100
μ	1	1	-
λ	1	1	-
Crossover probability (P_c)	-	-	0.5
Mutation probability (P_m)	0.1	0.1	0.05
Tournament size	-	-	2
Elitism	-	-	0.1
Evaluations	10,100	10,100	10,100

Table 2: Genome size for correlated input approaches

Approach	Input arguments	Intervals	Genome size
Basic	5	2	186
		3	1,089
	7	2	762
		3	9,837
Delta	5	2	124
		3	726
	7	2	508
		3	6,558

where the indirect approaches require a longer length corresponding to the detail described in Section 3. The approaches were used to seek values of the coefficients that maximise the execution time of polynomial solver for quartic and sextic equations. Accordingly, the genome sizes for the direct approaches are 5 and 7, whereas the genome sizes for the indirect approaches are summarised in Table 2.

4.3 Method

4.3.1 Direct Approaches. For the first part of the experiments, a particular metaheuristic technique was executed with the polynomial root-finding routine for ten trials, each of which was given a different seed taken from `random.org`. An execution time of a task, which is a fitness function, is captured in nanoseconds (ns) by using function `clock_gettime` with a clock source `CLOCK_MONOTONIC`. Also, in order to obtain a more precise fitness value of test vectors, each test case is repeatedly run with the task for 100 times and then a *median* of these runs is used to represent the fitness value. Although this approach is undoubtedly computationally intensive, it is a reasonable way to eliminating noise from the collected data. For proof of concept, we simply need a reliable cost function. Thus, such a median empirically determined measurement suffices. Other measures are not precluded.

4.3.2 Indirect Approaches. In the second part, after the metaheuristic algorithm constructs a tree, the tree will be used to generate ten test data samples. The sampled test vector that gives the highest execution time will be stored and its execution time will be used to represent the fitness value of the tree. The metaheuristic will then continue its search process to find more suitable trees, i.e. sets of intervals and probabilities. For our preliminary investigation a fixed delta value (Δ) was given at 10 for delta-based correlational approaches, whereas a randomised delta value (Δ_r) was sampled from a given range of 1 to 20.

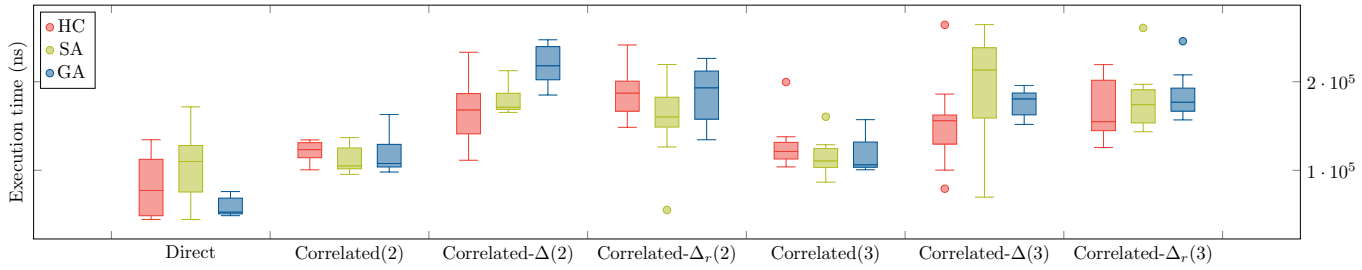
5 RESULTS AND DISCUSSIONS

The results from direct approaches are used as a baseline for assessing the effectiveness of dependent input strategies in finding extreme execution times of a polynomial root finder. The complete results of both experimental parts are illustrated in Figure 4. Each box depicts a distribution of the best fitness values obtained from ten trials of each approach to stress the polynomial solver with a particular number of input arguments, i.e. 5 and 7, respectively. Moreover, the box's whiskers indicate variability outside the upper and lower quartiles.

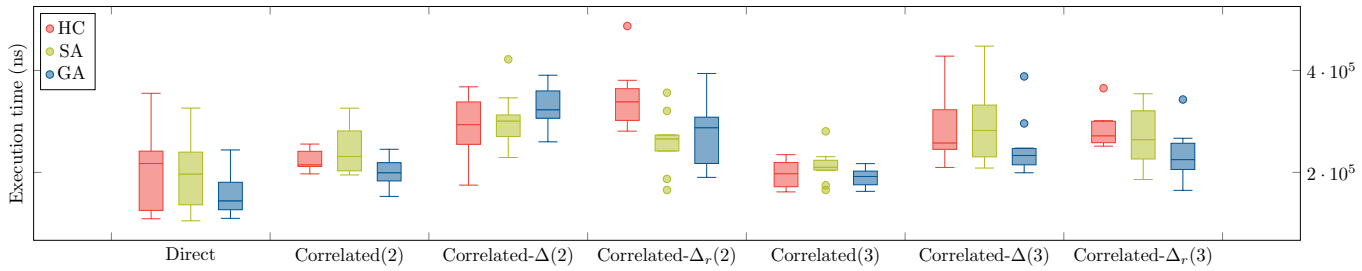
Besides, the differences between the initial and the final (best) fitness values gained from ten trials of the direct and indirect approaches are partially depicted in Figure 5. A darker bar represents an initial fitness, whereas a lighter bar represents the best fitness. Also, the percentage improvement is given on the tip of the lighter bar.

Furthermore, the longest execution time among ten trials of each approach on each particular input size is given in Table 3. Table 3 additionally includes the best among ten trials of randomised testing for the purpose of comparison as it is an important means of testing systems and is also an important benchmark for evaluating new approaches.

According to Figure 4, overall, the dependent input strategies performed effectively when the subdomain intervals are restricted with either a fixed delta value ($\Delta = 10$) or a randomised value

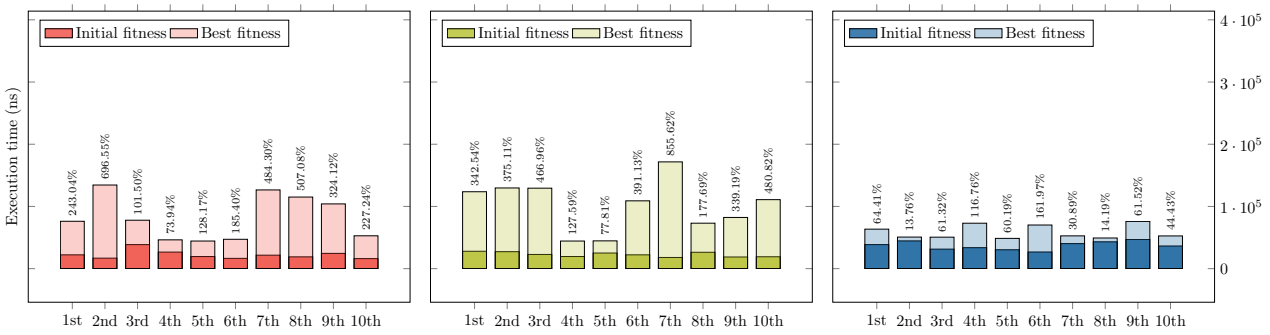


(a) Quartic equation (5 coefficients)

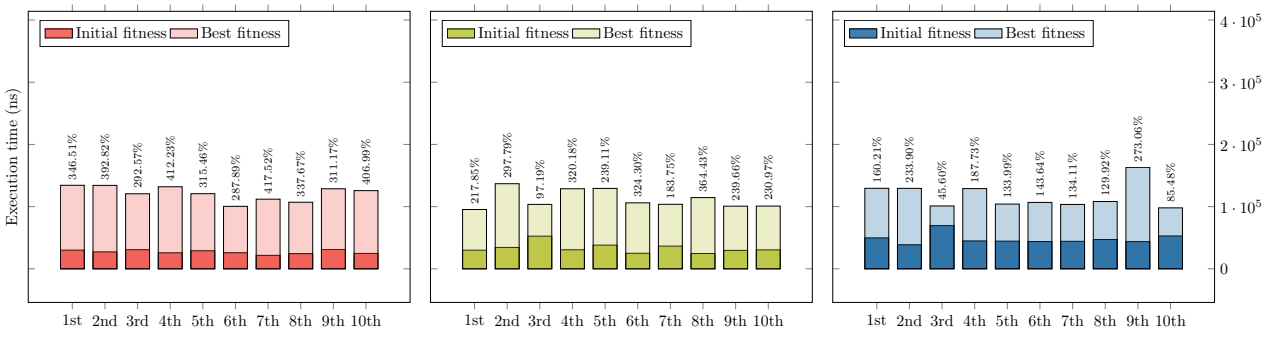


(b) Sextic equation (7 coefficients)

Figure 4: Distribution of execution times for each approach across 10 trials on the quartic and sextic equations



(a) Direct HC **(b) Direct SA** **(c) Direct GA**



(d) Correlated(2) HC **(e) Correlated(2) SA** **(f) Correlated(2) GA**

Figure 5: Results of a basic correlational method with 2 interval choices on the quartic equation (5 coefficients)

Table 3: A comparison of best fitness value of each algorithm on quartic and sextic equations

(a) Quartic equation (5 coefficients)

		2 interval choices			3 interval choices		
Algorithm	**Direct**	**Correlated**	**Correlated-Δ**	**Correlated-Δ_r**	**Correlated**	**Correlated-Δ**	**Correlated-Δ_r**
HC	134,449.0	134,300.0	233,315.5	241,541.0	199,659.0	264,310.0	219,380.0
SA	171,629.5	136,841.0	212,466.0	219,500.0	160,457.0	264,523.0	260,820.0
GA	75,917.0	163,110.0	247,400.0	226,421.0	157,159.0	195,888.0	245,780.0
RS	66,978.0	-	-	-	-	-	-

(b) Sextic equation (7 coefficients)

		2 interval choices			3 interval choices		
Algorithm	**Direct**	**Correlated**	**Correlated-Δ**	**Correlated-Δ_r**	**Correlated**	**Correlated-Δ**	**Correlated-Δ_r**
HC	355,198.0	255,600.0	367,868.0	487,240.0	234,817.0	428,145.0	365,160.0
SA	326,249.0	326,003.0	346,301.0	356,321.0	280,537.0	447,760.0	354,259.5
GA	243,980.0	245,364.0	390,689.5	394,097.0	217,162.0	388,320.0	343,003.0
RS	204,990.0	-	-	-	-	-	-

of delta ($\Delta_r = U(1, 20)$). Particularly, in comparison with direct approaches, the temporal test vectors indirectly generated from the dependent approaches delivered more extreme execution times in almost all cases; the basic method was inferior to the direct ones in some cases. For example, in case of SA on quartic equation (as shown in Figure 4a), although both basic dependent approaches, i.e. with two and three interval choices, were more stable over ten trials and have higher fitness values on average, the extreme execution time appeared on the direct SA. A similar situation also occurred in the cases of HC and SA on the sextic equation case (as presented in Figure 4b).

In addition, in terms of interval choices, the number of choices per each test input may increase chances of choosing appropriate subdomains for the dependent approaches; the more choices the better. However, in some cases, especially when the delta value is randomly given by a specific range, HC with two interval choices produced better results than the three interval options.

The effectiveness of such the dependent approaches is also able to reflect the ability of a metaheuristic on seeking subdomains that are likely to encompass inputs with extreme times. Regarding the results shown in Figure 4, single-solution based metaheuristics, i.e. HC and SA, effectively supported correlated input strategies to find test vectors that maximise the execution time of the polynomial solver in almost all of the cases. Even though GA was inferior to single-state approaches on searching for temporal test vectors in several cases, the execution times produced by indirect GA were more improvable than direct GA. In particular, as illustrated in Figure 5, a population-base metaheuristic generally started its initial fitness with a higher value compared with single-point metaheuristics as it gets more chance to select the best candidate solution among its population. However, at the end of the search process, direct GA delivered unremarkable best fitness value, unlike HC and SA, which delivered more desirable fitness values as shown in Figure 5c. For indirect GA, on the other hand, its final fitness values were considerably improved and closely resembled the indirect single-solution based metaheuristics as demonstrated in Figure 5f.

Table 4: The best values of coefficients

Input size	Values of coefficients
5	−21,492; −11,333; 26,663; 10,440 and −13,244
7	6,134; −16,035; 28,217; −16,510; −15,982; 32,398 and 17,391

On the whole, as listed in Table 3, SA and the randomised delta based correlational approach with three interval choices was the best for the quartic equation problem, whereas HC and the randomised delta based correlational approach with two interval choices was the best among all the experiments of sextic equation.

We verified the test inputs presented in Table 4 that provided the most extreme execution times for the quartic and sextic polynomials over all experiments by solely executing each extreme test vector with the polynomial solver on the P4080 for 100 times. As shown in Table 5, the verification proved that the correlated input strategies are able to effectively seek the coefficients that maximise the execution time of the polynomial root-finding for the quartic and sextic equations.

The results further prove our aforementioned assumption that extreme execution times may be concentrated in small partitions of the input space and reducing the input domain could increase the chances of inputs with extreme times being sampled. It could also be concluded that single-solution based metaheuristics were the most effective approaches for correlated input strategies, including basic an fixed delta based methods.

Table 5: Execution times of the best values of coefficients

Input size	N	Mean	Mdn	SD	min.	max.
5	100	265,800	264,600	4,817	264,579	297,859
7	100	487,800	486,700	3,945	486,659	520,146

6 CONCLUSIONS

This paper investigated the ability of our proposed approach, i.e. dependent input strategies, to seek temporal test vectors that maximise the execution time of a task running on a multicore chip. The dependent inputs approach aims to address the circumstance that test inputs that maximise the execution time may be concentrated in small parts of the input space and that good choices (as far as extreme execution times are concerned) for input parameters depend on the choices made for other input parameters. We presented an approach that reduces the input domain sampled, together with a sampling mechanism that selects the next test input parameter in a manner that depends on parameters chosen earlier.

We presented three different ways of setting an interval of a subdomain produced by the dependent approach: 1) basic method; 2) fixed delta based method; and 3) randomised delta based method. Overall, the results demonstrated that our dependent approach performed effectively when the interval of subdomains are restricted with either a fixed delta value or a randomised value of delta from a given range. Additionally the number of interval choices per each test input may increase chances of choosing appropriate subdomains for correlational approach, although in some cases, especially when the delta value is give by a specific range, the option of two interval choices produced better results than the three interval one.

Furthermore, the metaheuristic optimisation techniques are used to search for appropriate values of parameters to be assigned to such subdomains of the correlational approach. Therefore, the search space for metaheuristic searches are the space of parameterised distributions, not the space of inputs. Regarding the results, single-solution based metaheuristics, i.e. HC and SA, were effective in supporting correlated input strategies to find test vectors that maximise the execution time of the GSL's polynomial root-finder.

Several research directions are possible. For example, utilising other probability distributions, such as beta, binomial, normal, exponential and Weibull distributions, to generate test data from selected subdomains, as well as applying the proposed approach to other SUTs and also to other problem domains. The incorporation of further distributional flexibility should not present any significant problems. There is a wealth of parametric distributions available and a subdomain's distributional type and its parameters can simply be incorporated into the genome.

ACKNOWLEDGMENTS

The work is supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant No.: EP/J017515/1—Dynamic Adaptive Automated Software Engineering and by the Royal Thai Government.

REFERENCES

[1] Wasif Afzal, Richard Torkar, and Robert Feldt. 2009. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology* 51, 6 (2009), 957 – 976. https://doi.org/10.1016/j.infsof.2008.12.005

[2] Ilhem BoussaÃŕd, Julien Lepagnot, and Patrick Siarry. 2013. A survey on optimization metaheuristics. *Information Sciences* 237 (2013), 82 – 117. https://doi.org/10.1016/j.ins.2013.02.041 Prediction, Control and Diagnosis using Advanced Neural Computations.

[3] A. Burns and J. A. McDermid. 1994. Real-time safety-critical systems: analysis and synthesis. *Software Engineering Journal* 9, 6 (Nov 1994), 267–281. https://doi.org/10.1049/sej.1994.0036

[4] Gunter Dueck and Tobias Scheuer. 1990. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys.* 90, 1 (1990), 161 – 175. https://doi.org/10.1016/0021-9991(90)90201-B

[5] A. Engel. 2010. *Verification, Validation and Testing of Engineered Systems.* John Wiley & Sons. https://books.google.co.uk/books?id=H6N2CgAAQBAJ

[6] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich. 2015. *GNU Scientific Library Reference Manual* (third ed.). The GSL Team.

[7] Patrick Graydon and Iain Bate. 2014. Realistic safety cases for the timing of systems. *Comput. J.* 57, 5 (2014), 759–774.

[8] M. Harman, Y. Jia, and Y. Zhang. 2015. Achievements, Open Problems and Challenges for Search Based Software Testing. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST).* 1–12. https://doi.org/10.1109/ICST.2015.7102580

[9] Courtney E. Howard. 2012. Modern microprocessors: Robust, high-performance aerospace and defense systems harness the power of innovative microprocessors. (March 2012). http://www.militaryaerospace.com/articles/print/volume-23/issue-3/technology-focus/modern-microprocessors.html/ [Online; posted 1-March-2012].

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671 arXiv:http://science.sciencemag.org/content/220/4598/671.full.pdf

[11] M. Levy and T. M. Conte. 2009. Embedded Multicore Processors and Systems. *IEEE Micro* 29, 3 (May 2009), 7–9. https://doi.org/10.1109/MM.2009.41

[12] Sean Luke. 2013. *Essentials of Metaheuristics* (second ed.). Lulu. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[13] Sean Luke. 2017. ECJ 24 and 25: A Java-based Evolutionary Computation Research System. (2017). http://cs.gmu.edu/~eclab/projects/ecj/ [Online; accessed 21-October-2017].

[14] P. McMinn. 2011. Search-Based Software Testing: Past, Present and Future. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops.* 153–163. https://doi.org/10.1109/ICSTW.2011.100

[15] Matthew Patrick, Rob Alexander, Manuel Oriol, and John A. Clark. 2015. Subdomain-based test data generation. *Journal of Systems and Software* 103 (2015), 328 – 342. https://doi.org/10.1016/j.jss.2014.11.033

[16] Hartmut Pohlheim and Joachim Wegener. 1999. Testing the Temporal Behavior of Real-time Software Modules Using Extended Evolutionary Algorithms. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2 (GECCO'99).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1795–1795. http://dl.acm.org/citation.cfm?id=2934046.2934210

[17] Freescale semiconductor. 2011. *P4080 Development System User's Guide.* Technical Report.

[18] Freescale semiconductor. 2014. *Running AMP, SMP, or BMP Mode for Multicore Embedded Systems.* Technical Report.

[19] David R. White. 2012. Software review: the ECJ toolkit. *Genetic Programming and Evolvable Machines* 13, 1 (2012), 65–67. https://doi.org/10.1007/s10710-011-9148-z

[20] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. 2008. The Worst-case Execution-time Problem — Overview of Methods and Survey of Tools. *ACM Trans. Embed. Comput. Syst.* 7, 3, Article 36 (May 2008), 53 pages. https://doi.org/10.1145/1347375.1347389