Journal of Engineering Science and Technology
Vol. 15, No. 5 (2020) 3142 - 3161

GAMIPOG: A DETERMINISTIC GENETIC MULTI-PARAMETER-ORDER STRATEGY FOR THE GENERATION OF VARIABLE STRENGTH COVERING ARRAYS

M. I. YOUNIS

Department of Computer Engineering,
College of Engineering, University of Baghdad, Baghdad, Iraq
E-mail: younismi@coeng.uobaghdad.edu.iq

Abstract

This paper discusses the construction of a test data generation strategy for variable strength covering array (VSCA). The complexity of a VSCA strategy affects the scalability to support higher strength of coverage, the size of the generated test suites, and the execution-time significantly. There are many desired characteristics like the deterministic feature, supporting higher strength of coverage, manageable test size, less order of complexity, and fast execution-time. Implementing these conflict demands in a single strategy is very challenging task. Facing these challenges, this paper proposes a deterministic genetic multi-parameter-order strategy for generating VSCA called GAMIPOG. GAMIPOG combined the one-test-at-a-time, one-parameter-at-a-time, and meta-heuristics strategies to take advantage of them with a step-by-step example to illustrate the concept. Besides, this paper reviews the state-of-the-art of the VSCA strategies and provides a systematic analysis in a tabular form to discuss the desired features and the similarities and differences among VSCAs strategies. The practical results are so competitive as compared with the existing strategies in terms of the test size. Moreover, the GAMIPOG has an intermediate order of complexity, fast execution-time, and minimal test size in most cases. Finally, during running the experiments, new upper bounds for VSCAs have been reported by the GAMIPOG.

Keywords: Covering array, Combination testing, Genetic algorithm, Greedy algorithm, t-way testing, VSCA.

1. Introduction

The downscale of low-cost, efficient component systems enables the developers to adopt system integration. However, to ensure the reliability and the quality of these systems has increased the need for systematic testing. Combinatorial testing acts as an efficient black-box sampling strategy for generating test data for the system under test (SUT). The generation process adopts the t-way combinatorial interaction between parameters-values, (i.e., the values or levels for each parameter) [1-3].

Combinatorial testing has a colourful history involve the strategies for generating the test suite based on the strength of the coverage (named t-way). Most of the researchers in this area focused on obtaining minimal size for covering array (CA) of a certain t-way strength that covers the t-way tuples among parameters. The size of CA increments significantly increases in t, which makes the testing so expensive. However, researchers observed that not all parameters require higher strength of interaction among them during the testing. Thus, in a system which has some components equals to k, a technique is desired to build a CA for the strength of coverage equals to t that contains the t-way tuples between k parameters and also contains the ti-tuples (where ti > t) interactions among the subset of k parameters. The term variable strength covering array symbolized as VSCA (N; t, k, (v1, v2, vk), C) is used to notate such systems [2, 4, 5]. Where C is a subset of VSCA has a variable strength of testing. For instance, a VSCA (10; 2, 2^{11}, {CA (3, 2^3)^2}) is shown in Fig. 1.

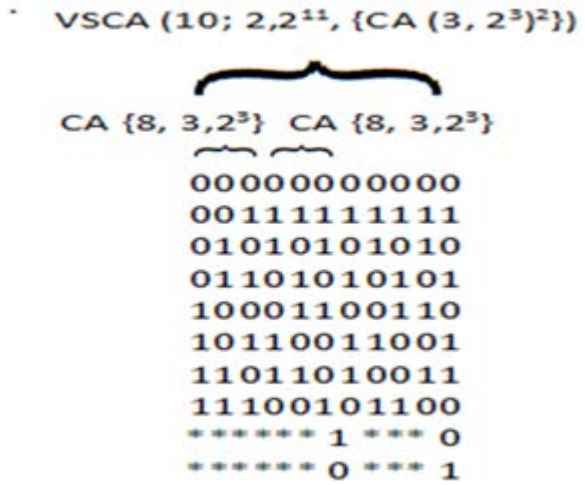

Fig. 1. The mathematical notations example for combinatorial VSCA (10; 2, 2^{11}, {CA (3, 2^3)^2).

In this VSCA, the size is ten test cases and covers all 2-way tuples interaction between the components. When there are *n* identical sub covering arrays where *t*, *k*, and *v* are fixed, the *CA*s can be symbolized as $CA\ (t, v^k)^n$. In our example, VSCA (10; 2, 2^{11}, {CA (3, 2^3)^2}) contains two sub-disjoint CAs, each of them contains the 3-way tuples among three parameters with two values for each parameter as

depicted in Fig. 1. The "*" symbol is known as "don't care" that can be assigned to any value (either 0 or 1).

Recent studies and surveys [1-3] have reported that there are more than 100 strategies exist in the literature. Most of these strategies generate CAs. However, only a limited number of these strategies owned the feature for generating the VSCAs. The VSCA strategies are extensively divided into computational-based and the search-based software testing (SBST) strategies. Computational algorithms use the greedy approach to build the VSCA by adopting either one-test-at-a-time (OTAT) [6-11] approach or one-parameter-at-a-time (OPAT) approach [8, 12, 13]. In the OTAT approach, the strategy adds one row to the test suite. While in the OPAT approach, the strategy adds one column to the test suite to cover the most tuples, followed by the OTAT approach if there are still more tuples. The SBST strategies use meta-heuristic techniques to find near minimal-size solutions for combinatorial testing [14-20]. Meta-heuristic techniques look for feasible solutions over a large set [16, 21]. Another classification is for the combinatorial strategies whether they are deterministic or non-deterministic. A non-deterministic strategy adopts a random search for parameter-values, (i.e., the values or levels for the parameter. For this reason, by running the algorithm multiple times, it generates sometimes minimal testing size). However, the practical testing requires to generate a deterministic test suite; especially, for fault identification and localization [3]. More recently, Moura et al. reported a new methodology to generate VSCA based on hyper graphs to identify the dependent events in theory [22]. One common feature among VSCA strategies is that they combine the priority of ti and t tuples during the tuples' space generation and the test suite generation. Consequently, this may lead to an over fitting problem, i.e., undesired computation.

Nevertheless, developing these algorithms it is considered as an NP-Complete problem [2, 3]. Furthermore, finding a minimal test size is considered as an NP-Hard problem [3]. The complexity of a VSCA strategy affects the scalability to support higher strength of coverage, the size of the generated test suites, and the execution-time significantly. There are many desired characteristics besides the deterministic feature, supporting higher strength of coverage, manageable test size, less order of complexity, and fast execution-time. Implementing these conflict demands in a single strategy is very challenging task. Facing these challenges, this paper proposes a deterministic genetic multi-parameter-order strategy for generating VSCA called GAMIPOG. The GAMIPOG is a genetic hybrid strategy that works in a step-wise-refinement approach by breaking the problem into sub-problems and finds a global solution by merging these solutions in iterative steps. In other words, the genetic layer performs domain reduction (simplifying the problem's domain to make it more solvable). Unlike existing strategies, the purpose of the genetic layer is to break the tie between the higher strength sub-arrays and the base strength by giving the priority to ti tuples to be covered in the first rows during the generation of the VSCA, i.e., at the beginning. The genetic hybrid layer consists of two algorithms: A Deterministic Multi-Parameter-Order Genetic Algorithm (GA) and an OPAT algorithm. The Modified Input Parameter Order General (MIPOG) [23-25] algorithm has been selected as a readymade component to facilitate generating the sub-solutions under the control of the genetic layer.

To ficitate the implementation, this work selects the Modified Input Parameter Order General (MIPOG) [23-25] algorithm as a readymade component to facilitate

the generation of the sub-solutions under the control of the genetic layer. In short, the MIPOG search for a possible extension by adding parameter-value in OPAT fashion followed by vertical expansion if necessary in OTAT fashion during the generating of the sub-solutions. Integrating these components altogether yields GAMIPOG (Genetic-Multi-Parameter-Order-Algorithm/MIPOG).

The organization for remaining sections for this paper is as follows. Section 2 describes the proposed GAMIPOG strategy with an illustrative example. Section 3 addresses the scope and the validation of the study. Also, this section discusses the trends and the complexities of the VSCA strategies in a tabular form. Section 4 describes the benchmarking experiments to evaluate the GAMIPOG strategy, compare it against other VSCA strategies, and state the contribution of this paper. Finally, Section 5 states the conclusion and gives some possible avenues for future research.

2. The GAMIPOG Strategy

The genetic process of the GAMIPOG strategy involves three phases:

- The initialization phase, the parents.
- Selection phase.
- The chromosomes' production (Population) phase.

In the initialization phase, the genetic layer orders the MIPOG algorithm to generate the search space by producing the Parents' lists and the desired features lists (Children). The parents' selection phase gives priority to the longest chromosomes, i.e., with the highest strength of coverage among the tuples to be the parents of the first generation in the population. After selecting the parents, the chromosome production is performed simply by taking the first genes from the first parent and search the most suitable genes from the second parent exhaustively that maximizing the coverage of the desired features, (i.e., calculate the fitness score). The fitness score is the number of uncovered t-way tuples inside the candidate chromosome. After finding the desired chromosome from the other parent(s), the genes are combined to produce the chromosome (the test case). Next, this chromosome is appended to the population's list (test suite). Finally, the covered genes are deleted from the parents' list as well as from the desired features lists, i.e., reduce the search space). These operations are iteratively done until the parents' lists are empty. Likewise, the parent selection is repeated and the Children with the highest chromosomes' length are selected to be the parents for the next generation. These operations are iteratively done until the termination condition is achieved, i.e., the search space is empty). After breaking the tie among the variable strength parameters, the MIPOG generates the test suite on the base strength for the remaining parameters. The GAMIPOG process is shown in Fig. 2.

For clarity and demonstration of the GAMIPOG strategy, we return to the example given in the previous section, to show how to generate the test suite and identify N for the VSCA (N; 2, 2^{11}, {CA (3, 2^3)^2}). To facilitate the illustration, assume further that the names of the parameters (the columns in Fig. 1); in order, are ABCDEFGHIJK respectively. Each parameter-value represents a gene. Each row represents a chromosome (test case).

```
1.  Algorithm Genetic Multi-Parameter Order {
    // Initialization phase
    //Splitting the domains
2.  Identify the Parents' strength of coverage (ti) and the base strength of
    coverage (t)
    // Building the search space
3.  Generate the Parents' lists by MIPOG, (i.e., CAs >t)
4.  Generate the Children's lists by MIPOG, (i.e., CAs =t)
5.   Let's ts be the test suite (Population list)
6.  While (Search space is not empty) {
    // Parents Selection Phase
7.  Select Parents' Lists that have the maximum chromosomes' length
  //Chromosome Production Phase
8.  While (First Parent's list is not empty) {
        a.  Select the first genes from the first Parent's list
        b.  Select the remaining genes from the other Parent(s)' list(s) by
            appending them to the first genes according to the fitness score.
        c.  Append the desired chromosome to ts.
        // Reduce Search Space
        d.  Delete the covered tuples from the Search Space
        e.  } // While
9.  }// While
    // Working for remaining parameters with base t
10. Initialize the MIPOG with ts
11. For each of the remaining parameters do the horizontal extension and the
    vertical expansion.
12. Return ts
13. }//Algorithm
```

Fig. 2. The GAMIPOG algorithm.

The genetic layer identifies two sub-CAs each with strength equals three and consists of three parameters with two values each. First the Genetic layer order the MIPOG to generate two sub-solutions ,(i.e., CA {8; 3, 2^3} and CA {8; 3, 2^3)) for the parameters ABC and DEF respectively, i.e., the 3-way parents' lists. These two sub-solutions act as two halves of chromosomes' that need to be combined in the genetic layer. Likewise, the MIPOG generates the features Children's lists (in our working example, the 2-way tuples for the parameters: AD, AE, AF, BD, BE, BF, CD, CE, and CF). Notably, the intertwined desired features, (i.e., (AB, AC, and BC), (DE, DF, and EF)) are not generated by the MIPOG because they are already covered by the first Parent's lists (ABC) and second Parent's lists (DEF), respectively. Thus, the MIPOG generates the search space (tuples) under the control of the genetic layer to prevent the MIPOG from generating undesired tuples. The Population list is empty. At this point, the initialization phase is finished. The selection phase selects ABC and DEF as first and second Parents, respectively (each of them has chromosome length equals 3) as depicted in the first half of Fig. 3.

The chromosome production phase starts by selecting the first genes, i.e., "000" from the first parent and searches iteratively the genes from the other parent, records the fitness score. The first candidate solution is "000000". The fitness score is nine because this candidate chromosome covers one tuple in each Child list (shown in red color). All other candidate chromosomes, (i.e., "000001", "000010", and "000111") have the same fitness score that is not greater than the first recorded score. Therefore, the first candidate is the desired chromosome (test case). The Genetic process appends the generated chromosome to the population list (test suite) and deletes the covered genes from the search space (Parents' and Childs' lists). Likewise, the first

genes, (i.e., "001") from the first parent are combined with the seventh genes, (i.e., "111") from the second parent to produce the desired chromosome, (i.e., "001111"), shown in green color, and so forth. Notably, the last two test cases selected in first-in-first-out fashion because all the 2-way tuples are covered in the previous iteration. Thus, the result of the genetic process is the population list witch is VSCA (8; 2, 2^6, {CA (3, 2^3)^2} as shown in the second half of Fig. 3.

The population size is eight (the first eight rows and the first six columns in Fig. 1). At this point, the genetic layer assists the MIPOG to deal with multi-parameter-order, i.e., performs the generation of the test suite in 6-parameters-at-a-time fashion. Whilst, the MIPOG assists the genetic layer by providing the search space. Instantly, the genetic layer passes the population list to the MIPOG. Considering the remaining parameters, i.e., GHIJK, the MIPOG continues in its normal operations in OPAT fashion with the based strength of coverage (t=2). For the parameter G, the MIPOG generates the 2-way tuples space, (i.e., AG, BG, CG, DG, EG, and FG) and searches for the most suitable gene-value (either 0 or 1) that has the maximum fitness score during the horizontal extension, appends the winner gene to test case, (i.e., extends the chromosome by appending the G's-value). After the horizontal extension, the tuples' space is empty. As such, there is no vertical expansion. the resulted population list is symbolized by VSCA (8; 2, 2^7, {CA (3, 2^3)^2}) (the first eight rows and the first seven columns in Fig. 1). Likewise, the size of the population's size invariants when extended by the parameters H, I, and J (the 8th till the 10th columns in Fig. 1). For the parameter K in the last iteration, after the horizontal extension, there are two non-covered tuples. These two tuples cannot be combined during the vertical expansion; therefore, these two tuples are added to the population list as two test cases in OTAT fashion (the last two rows in Fig. 1). The resulted VSCA (10; 2, 2^{11}, {CA (3, 2^3)^2}) is the desired test suite and N equals ten.

Fig. 3. The genetic process for constructing the VSCA (8; 2, 2^6, {CA (3, 2^3)^2}).

3. The Trends of the VSCA Strategies

The scope of this research is to consider the strategies that have the feature for generating the VSCA. These valuable strategies have different features as combinatorial minimization strategies. Manageable test size is one of the common interesting and competitive features. Fortunately, this desired feature is not a function of the running environment and execution-time and can be observed or taken from the literature. In contrast, the execution-time is a function of the complexity, the running environment, and the data structure. It was worth nothing to compare the running time from the literature. However, the complexity of these strategies is also important to be considered to estimate the running-time. In general, a deterministic strategy generates the test-suite in a single run and doesn't need to store the seeds or the test-suite, while a non-deterministic strategy requires that. Another important feature is the scalability of a higher strength of coverage for the interesting configurations now; we look at the interesting features and tabulated these features in Table 1. We start our discussion from the highest complexity and record the features during the interpretation.

Table 1. Summary of the trends of state-of-the-art VSCA strategies.

Features	SA [4]	r-GA [2]	ACS [17]	VSPSTG[18]	ABC [21]	HSS[20]	HABC [26]	GS [27]	MGS [28]	TVG	TSG [11]	PICT	ITCH	DARO [9]	DAFO [9]	ParaOrder [8]	ACTS	Desired for GAMIPOG
Complexity	Extremely High		Very High							High						Intermediate		
Parameter tuning	Required									Not required								
Deterministic/Non-deterministic	Non-deterministic									Deterministic								
Number of Run to record the best results	10	30	20	10	20 (t≤4) 5 (t>4)	30	20	10	30	Un-limited	1	1	1	1	1	1	1	1
Supporting Strength of Coverage for Sub-CAs	3	6	3	6	6	15	6	12	6	6	3	6	6	3	3	3	6	15
Priority of covering ti and t tuples	Same Priority																	ti first
Test size	Manageable											Undesired		Satisfactory		Manageable		

Examples of the SBST that support the generating of the VSCA are Simulated Annealing (SA) [4], Ant Colony Strategy (ACS) [17], Variable Strength Particle Swarm Test-suites Generation (VSPSTG) [18], Harmony Search Strategy (HSS) [20], Artificial Bee Colony (ABC) [21], the random Genetic Algorithm (r-GA) also called PairwiseGen [2], Hybrid Artificial Bee Colony (HABC) [26], tuned Genetic Strategy (GS) [27], and Modified Greedy Strategy (MGS) [28]. These strategies are non-deterministic. SBST strategies have many similarities. The objective of these strategies is to optimize the test size for the SUT. Unlike computational strategies, running the SA, r-GA, HSS, ABC, VSPSTG, and ACS requires tuning the parameters of the adopted strategy to avoid the local minimum during the test suite generation such that generated test-suite near minimum in size.

The SA starts with a random large array until finding an initial solution. Next, an exhaustive binary search is done to exchange the tuples based on the probability of occurrence to eliminate the weakest test case, (i.e., reduce the size by one). Iteratively, the SA repeats this process until finding a feasible solution. The best

solution has been recorded after 10 runs with distinct seeds [4]. After finding the base covering array, the SA tries to include the sub-array by doing the transformation of the SA algorithm, if not success, add the tuples and increment the size by one until covering all the tuples of sub-array(s). Thus, the approach in the implementation of the SA is to produce one-test-suite-at-a-time. The complexity of this implementation is extremely high. The SA dominant the minimum test size in most cases in the literature. However, the published results are restricted for small strength of coverage, i.e., $t \leq 3$.

The trend in the implementation of the r-GA strategy is to simplify the complexity of the SA a little bit to support the higher strength of coverage. The r-GA strategy starts from well-known size from the published results in the literature and adopting the genetic process. The genetic process in the r-GA implementation involves four steps, namely: initial population, selection, crossover, and mutation [2]. The initial population depends on the probability of the occurrence for each parameter-value. The size of the population is taken from the best well-known results. After the initialization of the population list, the multipoint cross-over selection for each chromosome is performed greedily randomly to increase the fitness and change the weak genes also randomly to gain more covered tuples. This process is repeated iteratively for all test cases until all tuples are covered, in this case, the r-GA strategy tries to reduce the size of the population by one and repeat the process until finding a more optimal solution if success. In short, the population list is the desired test suite. The best solution recorded after 30 runs with distinct seeds [2]. The complexity of this implementation is extremely high. Similar to r-GA, the GS is considered another variant of implementation of random Genetic algorithm with different parameters tuning. The tuned version of the GA is capable to achieve higher strength of coverage, up to $t = 12$ for the VSCA generation and requires 10 runs with distinct seeds [27]. In addition, the GS does not require knowing the desired size in advance. Thus, the complexity of this implementation is very high. On the other hand, the current implementation of the GS is supported the generation of VSCA when the sub-array has uniform values of parameters.

The trend in the implementations of HSS, MGS, ACS, HABC, VSPSTG, and ABC is to reduce the complexity further by adopting the OTAT approach. The best solution has been recorded after 30 [20], 30 [28], 20 [17], 20 [26], and 10 [18] runs with distinct seeds for HSS, MGS, ACS, HABC, and VSPSTG strategies, respectively. Regarding the ABC strategy, the best solution has been recorded after 20 runs for $t \leq 4$ and 5 runs for $t > 4$ [21]. Thus, the complexity of these implementations is very high. Like SA, ACS supports the generation of the VSCA for $t \leq 3$. The r-GA, MGS, VSPSTG, HABC, and ABC strategies support the generation of VSCAs up to $t \leq 6$. Whilst, the HS strategy is dominant the supporting of very high strength of coverage (up to $t = 15$).

Unlike SBTS strategies that need an initial feasible solution, computational strategies work in the reverse direction. They build the test suite from scratch. Moreover, they don't need parameters tuning. The strategies that adopt OTAT need to generate all the tuples to be covered before generating the test suite. For this reason, the complexity of these strategies is high. Examples of the strategies that adopted the OTAT approach are Pairwisely Independence Combinatorial Test data generator (PICT) [6], The IBM's Intelligent Test Case Handler (ITCH) [7], the Density Algorithm by Random Order (DA-RO) and the Density Algorithm by Fixing Order (DA-FO) [8, 9], the Test Vector Generator (TVG) [10], and Test Suite

Generator (TSG) [11]. The PICT, ITCH, DA-RO, DA-FO, and TSG strategies are deterministic. TVG is a non-deterministic tool. The PICT, ITCH, and TVG tools support generating VSCAs for $t \leq 6$. While, for TSG, DAFO, and DARO are supports generating VSCAs for $t \leq 3$. PICT is a valuable tool provided by Microsoft. The trend in the PICT implementation is to maximize the occurrence of the sub-CAs to be tested fully [6]. For this reason, the PICT tool has been reported to produce undesirable test size as a minimization strategy for generating VSCAs [2], [17-21]. Like PICT, ITCH has been reported to generate undesirable test size for generating VSCAs [18]. The Density Algorithm has been reported to generate satisfactory test size [20]. Both TSG and TVG strategies generate a manageable test suite [11].

The strategies that adopt the OPAT is approach for Generating VSCAs are Parameter Order (ParaOrder) [8] and Automated Combinatorial Test for Software (ACTS) [12, 13]. Both ParaOrder and ACTS strategies are deterministic. ACTS generates the VSCA using IPOG (Input Parameter Order General) option from the tool and supports the strength of coverage for $t \leq 6$. While, ParaOrder supports generating VSCAs for $t \leq 3$. Thus, the complexity of these implementations is intermediate. Notably, when the authors [2], [17-28] mentioned the execution-time in their environments, the OPAT strategies dominant the shortest execution-time significantly.

Now, we can discuss the trend in the proposed deterministic GAMIPOG strategy. Unlike the abovementioned VSCAs strategies, the genetic layer gives priority to the ti tuples to be generated and be covered first. The generation of tuples is controlled by the genetic layer to overcome the over fitting problem and to simplify the search space by eliminating the covered t-tuples during the initialization phase. The genetic layer also determines the number of parameters be covered by simply involve the parameters for ti tuples to be covered by the proposed deterministic multi-parameter-order GA in the OTAT approach. Since, the complexity of the OTAT is high, and to make the search space deterministic, the sub-CAs and the base t-tuples are generated by the deterministic MIPOG in OPAT fashion. Besides, the selection of ti-tuples is done by selecting the first ti-tuples from the first parent instead of searching the exhaustively first Parents' list. When generating the VSCA that contains both tuples, it is extended or even expanded by the MIPOG for the reaming parameters in the base strength. Unlike SBST, the GAMIPOG is a simple implementation of the genetic process aimed to have a deterministic feature. As such, the GAMIPOG has an intermediate level of complexity. In a nutshell, Table 1 gives a summary of the trends of state-of-the-art VSCA strategies and the desired features for the proposed GAMIPOG. To investigate more whether or not the desired features are implemented and to evaluate The GAMIPOG and compare it with the strategies we should adopt the same standard benchmarking experiments in the next section.

4. Results and Discussion

This section evaluates the GAMIPOG strategy and compares it against other computational strategies, (i.e., DA-FO, DA-RO, ParaOrder, TVG, TSG, and ACTS) and SBST strategies, (i.e., SA, r-GA, ACS, VSPSTG, ABC, and HSS). Both PICT and ITCH tools are excluded because they have been reported to produce undesirable test sizes. Four benchmarking experiments are conducted to support different configurations with higher strength of coverage. The first three

experiments are proposed in [4] and adopted also in [2, 8, 9, 11, 17, 21] and expanded in [18, 20] for some interesting configurations. Partially, these three experiments are adopted in [26-28]. While the forth experiment proposed in [18] and also expanded in [20] and adopted by some researchers. Among the abovementioned strategies, the TVG, and ACTS tools are available from the websites. Thus, they are run side-by-side with the GAMIPOG during the experimentations. Notably, these tools are implemented using the Java programming language. The running environment is a laptop with Windows 10 operating system, 8 GB of RAM, and 2.2 GHz CoreI7-2670QM CPU.

The objectives these four experiments to construct VSCAs, namely: VSCA (N; 2, 3^{15}, {C}), VSCA (N; 2, 3^{20}, 10^2, {C}), VSCA (N; 2, 4^3, 5^3, 6^2, {C}), and VSCA (N; 2, 10^1, 9^1, 8^1, 7^1, 6^1, 5^1, 4^1, 3^1, 2^1, {*C*}) and compare the generated test size generated by the VSCA strategies. We added the configuration {C=CA (6, 4^3, 5^3)} to the third experiment to study the behaviour of the GAMIOPG. The results of the other strategies are taken from the literature as indicated by its corresponding reference in the column. The results for running these experiments are tabulated in Tables 2 to 5. In these tables, the not available results are denoted by "NA" indicate that the test size is missing from the literature. The cells marked '–' indicate that the algorithm does not run this configuration. The dashed cells present a minimal test size to facilitate the comparison.

Now, we start our comparison with other strategies. At the glance, due to the NP-Hardness problem, there is no single strategy dominant minimal test size for all configurations Therefore, the minimal test size for the interesting configurations has been recorded one-by-one. A Question is normally raised why some strategies have the same generating test size?

What is interesting in these tables is that the size of the higher strength of coverage in the configurations, (i.e., the sub-arrays many-times drive the size of the generated test suite). This phenomenon can be observed in the following cases: referring to Table 2, the configurations CA (3, 3^3), CA (3, 3^3)^2, CA (3, 3^3)^3, and CA (3, 3^4) have test size equals to 27, (i.e., 3*3*3) which is quite sufficient to include the base arrays of strength 2 when increasing the number of parameters (marked by the dashed cells). A similar observation is also valid for the configurations CA (4, 3^4), CA (4, 3^5)^2, CA (5, 3^5), and the remaining configurations for ($t \geq 6$). For these test sizes the generated sizes are unreservedly minimal, i.e., can't be reduced further due to the mathematical prove. Such cases can be found in Table 3 for ($t \geq 4$), Table 4 for all dashed cells except the configurations Θ, CA (3, 4^3, 5^3, 6^1), and CA (3, 4^3, 5^3, 6^2), and Table 5 for all dashed cells except the configurations Θ and CA (3, 4^1, 3^1, 2^1). We dashed these configurations by green colour to facilitate further discussion. All these optimal, i.e., more manageable test suites are generated successfully by the GMIPOG except one in Table 4, i.e., C= CA (3, 4^3, 5^2) which acts as a counter-example and will be discussed later as far as the optimal results generation is concerned.

Referring to Table 2, for small strength of coverage, (i.e., $t < 4$), Both SA and r-GA generate minimal test size for t=2, (i.e., {C= Θ}) in the first row. As discussed previously, unlike other strategies, these strategies setup the desired size before the running of the experiments and the complexity of exhaustive search during the search is extremely high as mentioned previously. For this reason, it is expected to produce minimal results. Thus, both SA and r-GA outperform other strategies in

terms of test size. However, the other strategies (except the ParaOrder) produced a comparative result with less order of complexity. The result obtained from the ParaOrder is undesirable. This row signifies the requirement of an undesirable feature of extremely high computing which will be one avenue for future work to design an alternative algorithm.

Table 2. Comparison for test size for VSCA (N, 2, 3^{15}, {C}).

{C}	SA [4]	r-GA [2]	ACS [17]	VSPSTG[18]	ABC [21]	HSS [20]	HABC [26]	GS [27]	MGS [28]	TVG	TSG [11]	DARO [9]	DAFO [9]	ParaOrder [8]	ACTS	GAMIPOG
	Extremely High					Very High					High				Intermediate	
Θ	16	16	19	19	20	20	19	19	20	22	20	21	20	33	21	21
CA(3, 3^3)	27	27	27	27	27	27	27	28	27	27	27	28	29	27	27	27
CA(3, 3^3)^2	27	27	27	27	NA	27	NA	NA	NA	30	27	28	29	33	28	27
CA(3, 3^3)^3	27	27	27	27	NA	27	NA	NA	NA	30	28	28	30	33	29	27
CA(3, 3^4)	27	27	27	30	27	27	30	29	31	35	33	32	34	27	38	27
CA(3, 3^5)	33	33	38	38	33	38	39	38	39	41	40	40	42	45	41	39
CA(3, 3^4), CA(3, 3^5), CA(3, 3^6)	34	40	40	45	82	45	82	NA	NA	53	48	46	46	44	48	45
CA(3, 3^6)	34	40	45	45	45	45	45	46	44	48	48	46	46	49	48	45
CA(3, 3^7)	41	47	48	49	50	51	50	50	48	54	51	53	53	54	51	47
CA(3, 3^9)	50	57	57	57	58	60	50	57	57	62	59	60	60	62	63	57
CA(3, 3^{15})	67	74	76	77	81	77	81	75	81	81	82	70	78	82	83	78
CA(4, 3^4)	–	81	–	81	81	81	81	81	81	81	–	–	–	–	81	81
CA(4, 3^5)	–	91	–	97	90	94	93	92	96	103	–	–	–	–	100	81
CA(4, 3^7)	–	158	–	158	154	159	154	155	153	168	–	–	–	–	165	161
CA(5, 3^5)	–	243	–	243	243	243	243	243	NA	243	–	–	–	–	243	243
CA(5, 3^7)	–	441	–	441	446	441	439	441	NA	462	–	–	–	–	461	427
CA(6, 3^6)	–	729	–	729	729	729	729	729	729	729	–	–	–	–	729	729
CA(7, 3^7)	–	–	–	–	–	2187	–	2187	–	–	–	–	–	–	–	2187
CA(8, 3^8)	–	–	–	–	–	6561	–	6561	–	–	–	–	–	–	–	6561
CA(9, 3^9)	–	–	–	–	–	19683	–	19683	–	–	–	–	–	–	–	19683
CA(10, 3^{10})	–	–	–	–	–	59049	–	59049	–	–	–	–	–	–	–	59049
CA(11, 3^{11})	–	–	–	–	–	177147	–	177147	–	–	–	–	–	–	–	177147
CA(12, 3^{12})	–	–	–	–	–	531441	–	531441	–	–	–	–	–	–	–	531441
CA(13, 3^{13})	–	–	–	–	–	1594323	–	–	–	–	–	–	–	–	–	1594323
CA(14, 3^{14})	–	–	–	–	–	4782969	–	–	–	–	–	–	–	–	–	4782969

The second row is more interesting, (i.e., C= CA (3, 3^3)) that most of the strategies are produce the optimal results and GS, DARO, and DAFO strategies generate very competitive sizes, but seems to subject to local maximum when combined the priority of t and ti tuples during the generating VSCA. Here, we should make a deep argument when considering the complete performance based on both the test size and complexity. We can say that since the size is optimal, GAMIOPG, ACTS, and ParaOrder outperform both DARO and DAFO significantly, because GAMIOPG, ACTS, and ParaOrder have less test size with less order of complexity. TSG outperforms DARO and DAFO. While, GAMIOPG, ACTS, and ParaOrder outperform TSG, even though they are deterministic and produce the optimal test size, because of the order of complexity. Now, we can stress the NP-Hard problem more, by considering the complexity of comparison between deterministic and non-deterministic strategies. Since the deterministic strategies required only once run to produce the results, they outperform the non-deterministic strategies. For instance, it is unfair to compare both DARO and

DAFO performance with non-deterministic strategies without mentioning the complexity and the cost of running the experiments.

The TVG strategy outperforms both DARO and DAFO strategies under the cost of running the experiment for the same configuration 10 times. From this perspective, TSG outperforms TVG even though they have the same order of complexity. As such, GAMIOPG, ACTS, and ParaOrder outperform TVG significantly based on the order of complexity and non-deterministic feature for the TVG tool. More to the point, by looking at Table 6, we see that both GAMIOPG and ACTS outperform the TVG strategy significantly in the execution-time for a single run. While the total time for the TVG can be estimated to be ten times the recorded time. It is harder to compare the performance even between SBST strategies with a very high order of complexity. Besides they are non-deterministic strategies, they required parameters tuning for the search algorithm, different iterations, and different numbers of running, different initial solutions. When comparing the performance with extremely hard strategies, the difficulty increases because of these strategies required to estimate the desired size while other strategies built the test suite from scratch. Based on this perspective, we can say that GAMIOPG, ACTS, ParaOrder, TSG, and TVG outperform significantly both the strategies with a very high order of complexity ,i.e., MSG, HABC, HSS, ABC, VSPSTG, and ACS and with an extremely high order of complexity ,i.e., r-GA and SA. Likewise, the strategies with a very high order of complexity outperform both, r-GA and SA. Based on this argument we will discuss the remaining results to shortness the discussion especially for the configurations that have been highlighted by the green colour.

In the third and fourth rows, GAMIPOG outperforms the other strategies as far as the test size and overall all performance is concerned. Here, it seems both ACTS and ParaOrder are subject to a local minimum when giving the same priority to the mixing tuples. This point is valid in many cases that the GAMIPOG produces less test size.

Similarly, for the next row in Table 2, (i.e., C= CA (3, 3^4)). Both GAMIPOG and ParaOrder outperform other strategies as far as the performance and test size are concerned. In the next row, (i.e., C= CA (3, 3^5)), SA, r-GA, and ABC dominant the minimal test size. The ABC strategy outperforms SA and r-GA as far as the complexity is concerned. The other SBST, (i.e., HACS, GS, MGS, ACS, VSPSTG, HSS, and GAMIPOG) generate more comparable results and outperform the remaining computational strategies as far as to test size is concerned. Regarding the performance, GAMIPOG outperforms the other computational strategies as far as both the complexity and test size are concerned.

The configuration (CA (3, 3^4), CA (3, 3^5), CA (3, 3^6)) is very interesting. The SA strategy dominant the minimal test size while both the r-GA and ACS produce more competitive results. The HABC, ABC and TVG strategies produced undesirable test size for this configuration. The interesting point is that Para Order outperformed both ACTS and GAMIPOG in the term of generated test size. It is interesting to discuss this. In this case, the GAMIPOG generates the test suite in pure simplified OTAT, i.e., 15 parameters-at-a-times because there are three parents. A close look at the execution-time for this configuration we see it is near the time of when the CA (3, 3^{10}). As such, there are two design alternatives to minimize the test size for this case. The first one is to re-design and re-implement the MIPOG to deal with VSCA. The second design is to re-implement the genetic

layer to select two parents only at a time, i.e., give the superiority to the parameter order after the strength of coverage during the Parent selection phase. The interesting point is that the test size generated by the GMIPOG is outperformed by other computational OTAT strategies.

For the configurations (CA (3, 3^6), CA (3, 3^7)) the SA dominants the minimal test sizes while SBST strategies generate competitive results with a different order of complexity. In the case of (CA (3, 3^9)), both the HABC and the SA strategies dominant the minimal test size. Here, the HABC outperformed the SA as far as the overall complexity is concerned; the same observation is valid for the remaining SBST strategies. However, this observation cannot be generalized since the next row, i.e., CA (3, 3^{15}) the SA also generates minimal test size while DARO generates more competitive results and outperformed the remaining strategies. As discussed previously the GAMIPOG keeps the parameter order invariants when they have the same strength of coverage. For this reason, the result same as DAFO in this case. Addressing varying-parameter order is as interesting as an alternative design for future investigations.

It is clear that SA dominant the minimal test size for some configurations under the extremely high complexity cost. However, there is a lack of supporting higher strength of coverage for the valuable strategies as tabulated in Table 1. As such, we cannot compare the test sizes generated by them and our comparison is restricted to compare the remaining strategies. The GMIPOG dominant the unreservedly minimal sizes when generating the VSCA for the configuration CA (4, 3^5) and outperformed other strategies as far as the test size is concerned beyond the order of complexity. Notably, the other strategies may subject to a local minimum when generating the base tuples and combining the mixed coverage during the fitness functions computing. For the configurations (CA (4, 3^4), CA (5, 3^5), and CA (6, 3^6)) all the strategies that support the generation of VSCA for ($t>3$, $t<7$) are generated unreservedly minimal sizes under the different cost of complexity. The MGS strategy dominant the minimal test size for the configuration CA (4, 3^7) with competitive results from the other strategies. The GAMIPOG outperformed significantly other strategies as far as the test size is concerned for the configuration CA (5, 3^7).

Both GAMIPOG and HSS strategies, unlike other strategies, are scaled well to provide optimal sizes for $t > 6$ up to $t=15$, i.e., unreservedly minimal. While the GS strategy provide optimal sizes for $t > 6$ up to $t=12$. Unlike the valuable GS and HSS strategies which require a higher order of complexity, the GAMIPOG generates the test suites in the lower order of complexity with fast execution-time. This is stemmed from the fact that when increasing the strength of the coverage of the sub-CAs the more likely to provide sufficient space to include the base-array to provide optimal test-suites. This observation signifies the GAMIPOG approach. Unlike GS, both HSS and GAMIPOG strategies have excellent performance as optimization strategies for these cases and have been supported disjoint sub-array in the configuration. This is also the case for the results in Tables 3 to 6.

Referring to Table 3, the results of the ABC, HABC, GS, and MGS strategies are missing, as such, we cannot compare with them more. In the first row, The GAMIPOG, ParaOrder, DAFO, DARO, TSG, ACS, r-GA and SA strategies generate the same unreservedly minimal test size in different orders of complexity. While the other strategies are generating competitive results. In the second row the, DARO, TSG, ACS, r-GA and SA strategies generate the same unreservedly

minimal test size in different order of complexity. Also, other strategies are generating competitive results. From this observation, it is evident that a third design alternative is possible by making the deterministic GA generates the test suite in a pure OTAT manner. The third row is very interesting; SA dominant the minimal test size, GAMIPOG is more competitive than other strategies. The other strategies seem to have subjected to over fitting problems when combining the priority and may generate the intertwined tuples during the test case generating, and for this case, generate undesirable test size. We cannot compare with the other strategies that have not to support t >3. For (t > 3, t < 6) GAMIPOG, ACTS, TVG, HSS, VPSTG, and r-GA generate unreservedly minimal test size under the different cost of complexity. While both GAMIPOG and HSS generate unreservedly minimal test size for t > 6 under the different cost of complexity.

Sometimes the size of the base array is more than the size of the sub-CAs, this is also an interesting phenomenon that can be observed in the following cases: referring to Table 3, the configurations CA Θ, and CA (3, 3^{20}) have test size equals to 100 which can be determined from the base strength of the VSCA, i.e., 10*10). We can also find such cases in Table 4 for the configuration Θ and Table 5 for the configurations Θ and CA (3, 4^1, 3^1, 2^1). We dashed these configurations by red colour to distinguish them. Also, for these test sizes, the generated sizes are unreservedly minimal. For the other cases, we remain the cells of configurations without highlighting where the sizes more than the size of CAs but there is no mathematical prove due to the NP-hard problem, we see more verities in terms of test size for these sizes. In many of these cases, GMIPOG generates optimal results. Notably, that is the answer why the minimization strategies may share the same test size even though; they have different approaches and different implementations due to the aforementioned NP-Complete problem. Based on these observations, we will discuss the results.

Table 3. Comparison for test size for VSCA (N; 2, 3^{20}, 10^2, {C}).

{C}	SA [4]	r-GA [2]	ACS [17]	VSPSTG[18]	ABC [21]	HSS [20]	HABC [26]	GS [27]	MGS [28]	TVG	TSG [11]	DARO [9]	DAFO [9]	ParaOrder [8]	ACTS	GAMIPOG
	Extremely High					Very High					High				Intermediate	
Θ	100	100	100	102	NA	106	NA	NA	NA	101	100	100	100	100	102	100
CA(3, 3^{20})	100	100	100	105	NA	109	NA	NA	NA	103	100	100	105	103	102	107
CA(3,3^{20}, 10^2)	304	440	396	481	NA	450	NA	NA	NA	423	411	401	409	442	442	312
CA(4, 3^3, 10^1)	–	270	–	270	NA	270	NA	NA	NA	270	–	–	–	–	270	270
CA(5, 3^3, 10^2)	–	2700	–	2700	NA	2700	NA	NA	NA	2700	–	–	–	–	2700	2700
CA(6, 3^4, 10^2)	–	8100	–	8100	NA	8100	NA	NA	NA	8100	–	–	–	–	8100	8100
CA(7, 3^7)	–	–	–	–	–	2187	NA	NA	NA	–	–	–	–	–	–	2187
CA(8, 3^8)	–	–	–	–	–	6561	NA	NA	NA	–	–	–	–	–	–	6561
CA(9, 3^9)	–	–	–	–	–	19683	NA	NA	NA	–	–	–	–	–	–	19683
CA(10, 3^{10})	–	–	–	–	–	59049	NA	NA	NA	–	–	–	–	–	–	59049
CA(11, 3^{11})	–	–	–	–	–	177147	NA	NA	NA	–	–	–	–	–	–	177147
CA(12, 3^{12})	–	–	–	–	–	531441	NA	NA	NA	–	–	–	–	–	–	531441
CA(13, 3^{13})	–	–	–	–	–	1594323	NA	NA	NA	–	–	–	–	–	–	1594323
CA(14, 3^{14})	–	–	–	–	–	4782969	NA	NA	NA	–	–	–	–	–	–	4782969
CA(15, 3^{15})	–	-	–	–	–	14348907	NA	NA	NA	–	–	–	–	–	–	14348907

Referring to Table 4, the results of the GS strategy are missing, as such, we cannot compare with it more. In the first row (CA = (Θ)) the SA strategy dominant the unreservedly minimal test size while the r-GA and the TSG strategies provide more competitive results as far as the test size is concerned. All strategies generate competitive results for the configurations (CA (3, 4^3), (CA (3, 4^3), CA (3, 5^3)), and CA (3, 5^1, 6^2)), most of them generate unreservedly minimal size with a different order of complexity. For the remaining configurations (i.e., CA (3, 4^3, 5^2), CA (3, 4^3, 5^3, 6^1), and CA (3, 4^3, 5^3, 6^2)) the SA again dominant the minimal test size generation while both the TVG and the ParaOrder strategies generate undesired test size. For the configurations (CA (3, 4^3, 5^3, 6^2), and CA (3, 4^3, 5^3, 6^1)) both the ACS and the GAMIPOG strategies provide more competitive results to the SA. In the last configuration ,i.e., CA (3, 4^3, 5^3, 6^2) both the TSG and the GAMIPOG strategies provide more competitive results to the SA as far as the test size is concerned with a different order of complexity.

For (t > 3, t <=5) we continue our comparison for the strategies supporting these strength of coverage. The GMIPOG outperformed the other strategies because it generates generate unreservedly minimal size for all the interesting configurations. For the configurations (CA (4, 4^3, 5^2) and CA (5, 4^3, 5^3)) the GMIPOG strategy dominant the generate unreservedly minimal size and outperformed the other strategies significantly. For the other cases, all the strategies generate the same sizes with a different order of complexity. To study the behaviour of the GAMIPOG for (t=6) we added the configuration CA (6, 4^3, 5^3) to this standard experiment which is not been addressed by the other researchers. So, the result is restricted to the available tools. The GAMIPOG, ACTS, and TVG generate the unreservedly minimal size for this configuration. Lastly, for the configuration CA (7, 4^3, 5^3, 6^2) both GAMIPOG and HSS are scaled-well and have been generated the unreservedly minimal size for this configuration.

Table 4. Comparison for test size for VSCA (N; 2, 4^3, 5^3, 6^2, {C}).

{C}	SA [4]	r-GA [2]	ACS [17]	VSPSTG [18]	ABC [21]	HSS [20]	HABC [26]	GS [27]	MGS [28]	TVG	TSG [11]	DARO [9]	DAFO [9]	ParaOrder [8]	ACTS	GAMIPOG
	Extremely High				Very High							High			Intermediate	
Θ	36	37	41	42	44	42	42	NA	41	44	39	41	40	49	40	40
CA(3, 4^3)	64	64	64	64	64	64	64	NA	64	67	64	64	64	64	67	64
CA(3, 4^3, 5^2)	100	120	104	124	128	116	121	NA	120	132	125	131	132	141	132	108
CA(3, 5^3)	125	125	125	125	125	125	125	NA	125	125	125	125	125	126	126	125
CA(3, 4^3), CA(3, 5^3)	125	125	125	125	125	125	125	NA	NA	125	125	125	125	129	126	125
CA(3, 4^3, 5^3, 6^1)	171	212	201	206	213	212	212	NA	203	237	197	207	211	247	215	201
CA(3, 5^1, 6^2)	180	180	180	180	180	180	180	NA	180	180	180	180	180	180	180	180
CA(3, 4^3, 5^3, 6^2)	214	260	255	260	266	263	269	NA	258	302	239	256	261	307	263	243
CA(4, 4^3, 5^1)	–	320	–	320	320	320	320	NA	320	320	–	–	–	–	320	320
CA(4, 4^3, 5^1), CA(4, 5^2, 6^2)	–	900	–	900	900	900	900	NA	NA	900	–	–	–	–	900	900
CA(3, 4^3), CA(4,53, 6^1)	–	750	–	750	750	750	750	NA	NA	750	–	–	–	–	750	750
CA(4, 4^3, 5^2)	–	453	–	472	463	453	461	NA	461	496	–	–	–	–	479	400
CA(5, 4^3, 5^2)	–	1600	–	1600	1600	1600	1600	NA	NA	1600	–	–	–	–	1600	1600
CA(5, 4^3, 5^3)	–	2430	–	2430	2403	2430	2411	NA	NA	2583	–	–	–	–	2487	2000
CA(6, 4^3, 5^3)	–	NA	–	NA	NA	NA	NA	NA	NA	8000	–	–	–	–	8000	8000
CA(7,4^3, 5^3, 6^2)	–	–	–	–	–	48000	–	NA	–	–	–	–	–	–	–	48000

Referring to Table 5, the GMIPOG outperformed the other strategies in the sense that it always generates the unreservedly minimal sizes for these configurations. Moreover, the GAMIPOG dominant the minimal test size for the configuration CA

$(3, 10^1, 9^1, 8^1, 7^1)$ and outperform the other strategies significantly while both HSS and r-GA produced a more competitive size than other strategies for this case.

Table 5. Comparison for test size for VSCA (N; $2,10^1, 9^1, 8^1, 7^1, 6^1, 5^1, 4^1, 3^1, 2^1$, {$C$}).

{C}	SA [4]	r-GA [2]	ACS [17]	VSPSTG[18]	ABC [21]	HSS [20]	HABC [26]	GS [27]	MGS [28]	TVG	TSG [11]	DARO [9]	DAFO [9]	ParaOrder[8]	ACTS	GAMIPOG
	Extremely High				Very High						High				Intermediate	
Θ	NA	92	NA	97	NA	94	NA	NA	NA	99	NA	NA	NA	NA	90	90
CA(3, 10^1 9^1 8^1)	NA	720	NA	720	NA	720	NA	NA	NA	720	NA	NA	NA	NA	720	720
CA(3, 7^1, 61, 5^1)	NA	210	NA	210	NA	210	NA	NA	NA	210	NA	NA	NA	NA	211	210
CA(3, 4^1, 31, 2^1)	NA	92	NA	97	NA	94	NA	NA	NA	99	NA	NA	NA	NA	90	90
CA(3, 10^1, 9^1, 8^1, 7^1)	NA	740	NA	742	NA	740	NA	NA	NA	784	NA	NA	NA	NA	772	**720**
CA(3, 10^1, 9^1, 8^1), CA(3, 7^1, 6^1, 5^1)	NA	720	NA	720	NA	720	NA	NA	NA	720	NA	NA	NA	NA	720	720
CA(3, 10^1, 9^1, 8^1), CA(3, 7^1, 6^1, 5^1), CA(3, 4^1, 3^1, 2^1)	NA	720	NA	720	NA	720	NA	NA	NA	720	NA	NA	NA	NA	720	720
CA(4, 5^1, 4^1, 3^1, 2^1)	–	120	–	120	–	120	NA	NA	NA	123	–	–	–	–	142	120
CA(5, 10^1, 9^1, 4^1, 3^1, 2^1)	–	2160	–	2160	–	2160	NA	NA	NA	2160	–	–	–	–	2160	2160
CA(6, 7^1, 6^1, 5^1, 4^1, 3^1, 2^1)	–	5040	–	5040	–	5040	NA	NA	NA	5040	–	–	–	–	5043	5040
CA(7, 8^1, 7^1, 6^1, 5^1, 4^1, 3^1, 2^1)	–	–	–	–	–	40320	NA	NA	NA	–	–	–	–	–	–	40320
CA(8, 9^1, 8^1, 7^1, 6^1, 5^1, 4^1, 3^1, 2^1)	–	–	–	–	–	362880	NA	NA	NA	–	–	–	–	–	–	362880

Regarding the execution-time, referring to Table 6, since the GMIPOG, ACTS, and TVG strategies have been implemented using Java programming language, and have been running on the same computer. However, they have a different data structure and programming methodology. We can observe the execution-time and stress the need for the deterministic feature. It clear both ACTS and GAMIPOG strategies outperformed the TVG strategy significantly as far as the execution-time is concerned. Moreover, both of them generate the test size in a single run while due to the non-deterministic feature for the TVG strategy it requires to run 10 times. As such, the total execution time for the TVG is about 10 times the recorded time for it. What is interesting in this table, our approach may lead not just to optimize the test size, it often reduces the execution time. When selecting the parents, and don't generate the intertwined features that already covered, the execution-time is lower than the base time. For instance, the time required to generate the base array, i.e., C= Θ is more than the time required for generating the VSCA for the configuration CA $(3, 3^3)^2$. However, when the number of parameters increased for the GA algorithm, it may lead to more execution-time as discussed previously. Overall, both ACTS and GAMIPOG have very competitive execution-time.

In general, the GAMIPOG strategy unlike other minimization strategies, give the priority to the ti-tuple to be included generated and included first. Unlike other SBST strategies, it is deterministic and does not require parameters tuning with an intermediate level of complexity. The GAMIPOG strategy like other SBST strategies outperformed other computational strategies in most cases as far as the test size is concerned and provides competitive results. Like the ACTS tool, dominant the fast execution-time feature. Like the HSS and GS strategies, the GMIPOG strategy supports generating the test suite for high strength of coverage. Finally, the GAMIPOG contributes to find six minimal size for VSCA, namely: VSCA (81; 2, 3^{15}, {CA (4, 3^5)}), VSCA (427; 2, 3^{15}, {CA (5, 3^7)}) from Table 2,

VSCA (400; 2, 4^3, 5^3, 6^2, {CA (4, 4^3, 5^2)}), (2000; 2, 4^3, 5^3, 6^2, {CA (5, 4^3, 5^3)}), (8000; 2, 4^3. 5^3, 6^2, {CA (6, 4^3, 5^3)}), from Table 4, and VSCA (720; 2, 10^1, 9^1, 8^1 7^1, 6^1, 5^1, 4^1, 3^1, 2^1, {CA (3,10^1 9^1 $8^1$$7^1$)}) from Table 5. These new upper bounds for VSCA are shown in bold font in their corresponding cells.

Table 6. Execution-time (in seconds) for generating VSCA (N; 2, 3^{15}, {C}).

{C}	TVG	ACTS	GAMIPOG
Ɵ	0.047	0.016	0.018
CA(3, 3^3)	0.061	0.008	0.017
CA(3, 3^3)^2	0.064	0.021	0.017
CA(3, 3^3)^3	0.061	0.021	0.017
CA(3, 3^4)	0.078	0.011	0.009
CA(3, 3^5)	0.091	0.024	0.020
CA(3, 3^4), CA(3, 3^5), CA(3, 3^6)	0.236	0.009	0.064
CA(3, 3^6)	0.139	0.012	0.016
CA(3, 3^7)	0.153	0.023	0.031
CA(3, 3^9)	0.298	0.021	0.027
CA(3, 3^{15})	0.532	0.051	0.063
CA(4, 3^4)	2.01	0.019	0.016
CA(4, 3^5)	0.19	0.012	0.017
CA(4, 3^7)	0.78	0.014	0.024
CA(5, 3^5)	0.61	0.016	0.025
CA(5, 3^7)	3.6	0.054	0.049
CA(6, 3^6)	1.34	0.091	0.058
CA(7, 3^7)	-	-	0.061
CA(8, 3^8)	-	-	0.062
CA(9, 3^9)	-	-	0.063
CA(10, 3^{10})	-	-	0.064
CA(11, 3^{11})	-	-	0.074
CA(12, 3^{12})	-	-	0.077
CA(13, 3^{13})	-	-	0.083
CA(14, 3^{14})	-	-	0.093

5. Conclusions

This paper has been presented and evaluated a hybrid strategy called GAMIPOG. The GAMIPOG is a novel deterministic search-based-minimization strategy with an intermediate order of complexity and scales-well to support higher strength of coverage with fast execution time. Besides, this paper has been reported the trends of the VSCA strategies and mentioned the similarities and differences between the GAMIPOG and other strategies. The genetic layer consists of two algorithms: A Deterministic Multi-Parameter-Order Genetic Algorithm (GA) and the Modified Input Parameter Order General (MIPOG). The GA algorithm is a tuneless deterministic search-based algorithm that gives priority to the tuples during the test case generation. The experiments conducted show that the investigations directed demonstrate that the GAMIPOG, in most cases, outperforms the existing strategies in terms of VSCA sizes (except SA for small strength, (i.e., t < =3). Besides, the practical results reported outcomes of new upper bounds for VSCA. Again, the GAMIPOG hits many faces for the same coin.

During the evaluation, some results are not optimal as far as the test size is concerned and some design alternatives have been discussed. As such, there are multiple avenues for future research. One avenue is to re-design and re-implement the MIPOG to support the VSCA. Another avenue is to make different design

alternatives to select the parent's out-of-order, based on the density of the parameter-values and the strength of coverage, compare these variants in terms of complexity and size. Of course, under more cost of complexity, one of the target avenues is to study variant deterministic opportunities to investigate the genetic layer to make global optimization of the generated test suites in this paper by considering the non-minimal results as an initial population to be minimized in one-test-suite-at-a-time approach.

.

Nomenclatures

C	Configuration
k	Number of components (parameters)
n	Number of an identical sub covering arrays
N	Size (number of rows in a covering array)
t	Strength of coverage
ti	ti-tuples
ts	Test set
v	Number of levels (values)

Greek Symbols

Θ	Phi, empty set.

Abbreviations

ABC	Artificial Bee Colony
ACS	Ant Colony Strategy
ACTS	Automated Combinatorial Test for Software
CA	Covering Array
DA	Density Algorithm
DA-RO	Density Algorithm by Random Order
DA-FO	Density Algorithm by Fixing Order
GA	Genetic Algorithm
GAMIPOG	Genetic Multi-Parameter-Order-Algorithm/MIPOG
GS	Genetic Strategy
HABC	Hybrid Artificial Bee Colony
HSS	Harmony Search Strategy
IPOG	Input Parameter Order General
ITCH	IBM's Intelligent Test Case Handler
MGS	Modified Greedy Strategy
MIPOG	Modified Input Parameter Order General
NP	Non-deterministic Polynomial
OPAT	One-Parameter-At-A-Time
OTAT	One-Test-At-A-Time
ParaOrder	Parameter Order
PICT	Pairwisely Independence Combinatorial Test Data Generator
PSO	Particle Swarm Optimization
r-GA	Random Genetic Algorithm
SA	Simulated Annealing
SBST	Search-Based Software Testing
SUT	System Under Test

TSG	Test Suite Generator
TVG	Test Vector Generator
VSCA	Variable Strength Coverage Array
VSPSTG	Variable Strength Particle Swarm Test-suites Generation

References

1. Khalsa, S.K.; and Labich, Y. (2014). An orchestrated survey of available algorithms and tools for combinatorial testing. 2014 *IEEE 25th International Symposium on Software Reliability Engineering (ISSRE)*. Naples, Italy, 323-334.
2. Bansal, P.; Sabharwal, S.; Mittal, N.; and Arora, S. (2015). Construction of variable strength covering array for combinatorial testing using a greedy approach to genetic algorithm. *E-Informatica Software Engineering Journal*, 9(1), 87-105.
3. Younis, M.I. (2019). MVSCA: Multi-valued sequence covering array. *Journal of Engineering*, 25(11), 82-91.
4. Cohen, M.B.; Gibbons, P.B.; Mugridge, W.B.; Colbourn, C.J.; and Collofello, J.S. (2003) A variable strength interaction testing of components. *Proceedings of the 27th Annual International Conference on Computer Software and Applications*, 413-418.
5. Raaphorst, S.; Moura, L.; and Stevens, B. (2018). Variable strength covering arrays. *Journal of Combinatorial Design*, 26(9), 417-438.
6. Czerwonka, J. (2006) Pairwise testing in real world: practical extensions to test case generator. *Proceedings of 24th Pacific Northwest Software Quality Conference*, 419-430.
7. Hartman, A.; Klinger, T.; and Raskin, L. (2005). WHITCH: IBM intelligent test configuration handler. *Technical Report, IBM and Watson Research Laboratories.*
8. Wang, Z.; Xu, B.; and Nie, C. (2008). Greedy heuristic algorithms to generate variable strength combinatorial test suite. *The Eighth International Conference on Quality Software*, 155-160.
9. Wang, Z.; and He, H. (2013). Generating variable strength covering array for combinatorial software testing with greedy strategy. *Journal of Software*, 8(12), 3173-3181.
10. Arshem, J. (2019). TVG download web page. Retrieved December 25, 2019, from http://sourceforge.net/projects/tvg.
11. Abdullah, S.A.; Soh, Z.H.; and Zamli, K.Z. (2013). Variable strength interaction for t-way test generation strategy. *International Journal of Advances in Soft Computing and Its Applications*, 5(3), 65-74.
12. Forbes, M.; Lawrence, J.; Lei, Y.; Kacker, R.N.; and Kuhn, D.R.(2008). Refining the in-parameter-order strategy for constructing covering arrays. *Journal of Research of the National Institute of Standards and Technology*, 113(5), 287-297.
13. Yu, L.; Lei, Y.; Kacker, R.N.; and Kuhn D.R. (2013). ACTS: a combinatorial test generation tool. *IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST)*, 370-375.
14. Ali, S.; Briand, L.C.; Hemmati, H.; and Panesar, W.R.K. (2010). A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering*, 36(6), 742-762.

15. Garvin, B.J.; Cohen, M.B.; and Dwyer, M.B. (2011). Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering*. 16(1), 61-102.
16. Blum, C.; and Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, 35(3), 268-308.
17. Chen, X.; Gu, Q.; Li, A.; and Chen, D. (2009). Variable strength interaction testing with an ant colony system approach. *The APSEC'09. Asia-Pacific Software Engineering Conference*, 160-167.
18. Ahmed, B.S.; and Zamli K.Z. (2011). A variable strength interaction test suites generation strategy using particle swarm optimization. *Journal of Systems and Software*, 84(12), 2171-2185.
19. Cai, L.; Zhang, Y.; and Ji, W. (2018). Variable strength combinatorial test data generation using enhanced bird swarm algorithm. *IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 391-398.
20. Alsewari, A.R.A. ; and Zamli, K.Z. (2012). Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support. *Information and Software Technology*, 54(6), 553-568.
21. Alazzawi, A.K.; Rais, H.M.; and Basri, S. (2019). ABCVS: an artificial bee colony for generating variable t-way test sets. *International Journal of Advanced Computer Science and Applications*, 10(4), 259-274.
22. Moura, L.; Raaphorst, S.; and Stevens, B. (2019). Upper bounds on the sizes of variable strength covering arrays using the lovasz local lemma. *Theoretical Computer Science*, 800, 146-154.
23. Younis, M.I. (2011). *MIPOG: a parallel t-way minimization strategy for combinatorial testing.* Ph.D. Thesis, School of Electrical and Electronics Engineering, Universiti Sains Malaysia, Penang, Malaysia.
24. Younis, M.I.; and Zamli, K.Z. (2010). MC-MIPOG: a parallel t-way test generation strategy for multicore systems. *ETRI Journal*, 32(1), 73-83.
25. Younis, M.I.; and Zamli, K.Z. (2011). MIPOG an efficient t-way minimization strategy for combinatorial testing. *International Journal of Computer Theory and Engineering (IJCTE)*, 3(3), 388-397.
26. Alazzawi, A.K.; Rais, H.M.; and Basri, S. (2020). HABC: hybrid artificial bee colony for generating variable t-way test sets. *Journal of Engineering Science and Technology* (*JESTEC*), 15(2), 746-767.
27. Esfandyari, S.; and Rafe, V. (2018). A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy. *Information and Software Technology*, 94, 165-185.
28. Homaid, A.A.B.A.; Alsweari, A.A.; Zamli, K.Z.; and Alsariera, Y.A. (2018). Adapting the elitism on the greedy algorithm for variable strength combinatorial test cases generation. *IET Software*, 13(4), 286-294.