Preserving Inter-gene Relations During Test Case Generation using Intelligent Evolutionary Operators

Dimitri Michel Stallenberg
d.m.stallenberg@student.tudelft.nl
Delft University of Technology
The Netherlands

Mitchell Olsthoorn
m.j.g.olsthoorn@tudelft.nl
Delft University of Technology
The Netherlands

Annibale Panichella
a.panichella@tudelft.nl
Delft University of Technology
The Netherlands

ABSTRACT

Randomized variational operators can be very disruptive to the search process, especially when there exist dependencies between the variables under search. Within test-cases, these dependencies exist as well. This makes it interesting to evaluate the benefits of preserving these dependencies during test-case generation.

In this paper, we propose two variants of the Many-Objective Sorting Algorithm (MOSA). The first of which is based on Agglomerative Clustering, ACMOSA. The second is a Gene-pool Optimal Mixing based variant, GOMOSA. ACMOSA and GOMOSA model the inter-gene dependencies and use that model to intelligently perform crossover while preserving key building blocks within individuals. These novel techniques are evaluated in an empirical study and compared to MOSA and the Many Independent Objective algorithm (MIO). This study is composed of several benchmark RESTful APIs for which the algorithms generate test-cases.

The results of the empirical study show that, for 40% of the tested APIs, the novel techniques provide a significant benefit time-wise. For another 40% of the APIs, they perform equally well, and for 20% of the APIs under evaluation they performed worse.

KEYWORDS

Intelligent Evolutionary Operators, Search-based Software Engineering, Test Case Generation

1 INTRODUCTION

Manually writing test-cases can be a labor-intensive process. Extensive research is done to automate the process of generating test-cases. Evolutionary Algorithms (EAs) are often used to accomplish this goal. However, for large industry-scale applications with millions of lines of code, the process of automatically generating test-cases using EAs can still be slow and require a large amount of computational power. Additionally, the quality of the generated test-cases is often non-ideal as simple unit-level tests are often less meaning-full than system-level tests which represent a more realistic user interaction.

EAs rely on randomized mutation and crossover operators to generate and evolve solutions (chromosomes) for a given problem. In nature, this is not the case since chromosomes are recombined such that certain patterns in the chromosome that are essential to a species remain intact, i.e. certain genes of the chromosomes do not function properly without certain other genes. According to the building block hypothesis, the same inter-gene relations exist in computational problems [8]. Thus the usage of randomized variation operators can be extremely disruptive to the search process. More intelligent variation operators could guide the search for an optimal solution by preserving the inter-gene relationships. However, identifying such relationships can be very challenging for large datasets with many unknown variables.

Various researchers have addressed the disruptiveness of randomized variation operators. For example, Smith and Fogarty found that it can be beneficial to use self-adapting mutation rates that slow the mutation rate down when close to a solution [12]. By doing so there is a higher chance of preserving important building blocks. Other researchers used a more complex approach, e.g. Dioşan and Oltean used Genetic Programming to evolve better crossover operators [7]. However, these solutions lead to semi-static operators which still use randomness as their main guide. More intelligent solutions exist. According to Kim et al. the aforementioned inter-gene dependencies in nature can be modeled by a Bayesian Network (BN) [9]. This approach can also be used for artificial genes in computational problems. To find the appropriate network a Machine Learning (ML) approach can be effective. For example, Orphanou et al. proposed that the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) can be used effectively to model a BN [10].

The previously mentioned GOMEA method has been proven to be very effective in finding a proper BN for the benchmark problems mentioned by Orphanou et al. [10]. However, in these benchmark problems, the dependencies between some variables are obvious in the sense that they are strongly correlated. In real-world problems, this might not be the case. The method should be evaluated for practical problems, e.g. by modeling inter-gene relations in solutions of EAs.

This article focuses on identifying the inter-gene relations in generated test-cases. The test case will be generated by EvoMaster, a tool which uses EAs to generate system-level test-cases for RESTful API's [1]. The research questions for this article read:

- *What is the impact of intelligent evolutionary operators on EvoMaster?*
- *How does this impact translate to EvoMaster's performance in covering test targets?*

In this research, EvoMaster is extended with a model-based crossover operator that uses ML techniques to extract a model from a population of solutions. The first ML technique that will be evaluated uses Agglomerative Clustering (AC) to extract the aforementioned building blocks from the population. These building blocks can then be used during crossover. The second ML technique is somewhat more complex and uses GOMEA coupled with AC to find a BN representing the inter-gene relations. The BN is then translated to the building blocks. The GOMEA technique is based on the work of Orphanou et al. [10]. Both techniques will be evaluated by comparing the effectiveness of EvoMaster in generating code coverage for RESTful APIs and the efficiency of this process in terms of the used time.

2 BACKGROUND

For this study, EvoMaster will be used as the baseline tool for generating test-cases. EvoMaster will be modified to support the novel techniques proposed in this article. Both of these techniques will be evaluated on white-box problems, which means that they will have access to the number of covered targets among other statistics. To evaluate the techniques several benchmark RESTful APIs have been used. Namely, NCS, SCS, Features-service, Scout-API, and Proxyprint-kitchen. These APIs are part of the EvoMaster Benchmark project. The techniques will be evaluated on the number of targets they can cover in a certain amount of time and/or generations.

At the time of writing this article EvoMaster provides two algorithms. The first algorithm is the Many Independent Objective (MIO) Algorithm proposed by Arcuri [2]. The second algorithm is a variant of the Many-Objective Sorting Algorithm (MOSA) proposed by Panichella et al. [11]. Both of these algorithms are specifically designed for test-suite generation. To properly evaluated and compare the novel techniques ACMOSA and GOMOSA, the results of the MIO and MOSA are also reported. These results will answer whether preserving inter-gene relations is useful in search during test-case generation. Finally, due to the increase in time complexity the intelligent crossover causes, the novel techniques could turn out to be less effective than the original MOSA technique depending on the size of the RESTful API in terms of possible HTTP requests.

Crossover. It must be noted that the variant of MOSA implemented in the EvoMaster tool does not use the crossover operation. This variant of MOSA differs from the original algorithm proposed by Panichella et al. and will be referred to as VMOSA. However, to be able to properly evaluate the effectiveness of model-based variational operators, the original MOSA with One-Point crossover has been added to EvoMaster. As this study revolves around building block preservation, One-Point crossover is a logical choice as it is often used in schema (building block) oriented EAs. Finally, it is suspected that crossover has been removed from the algorithms in EvoMaster due to the disruptive consequences it has on searching for test-cases.

Gene Types. Solutions within the search algorithms of EvoMaster contain two types of genes. These genes describe what the final test-case will be like. The first type is the action gene which dictates the structure and order of steps in the test-case. For example, for RESTful API testing the action genes contain the HTTP requests that are available according to the provided swagger API documentation, e.g. "POST /authentication". The second type of gene is the input gene which form the inputs for the HTTP requests, e.g. the username and password. Because string-based input genes have way more variability than action genes, we suspect that action genes are more likely to be dependent on each other. For example, one API might have 20 possible actions of which 1 requires an input of length 10. This means that there are 26^{10} possible input genes (assuming alphabetic inputs only) compared to only 20 possible action genes. So even for this simplistic example, the difference in gene possibilities is staggering making it much more likely to find inter-gene relations between action genes than between input genes. For this reason, the focus of this research will be on the action genes.

Model-based Evolutionary Algorithms (MBEAs). Traditional EAs mostly use fixed heuristic rules for their crossover, mutation, and selection operators. These fixed rules limit the EAs in their capability to exploit partial solutions, also known as building blocks. MBEAs solve this problem by learning which parts of a solution are optimal. For example, as mentioned in the introduction, Optimal Mixing EAs like GOMEA are able to efficiently learn which parts of the solutions are important building blocks.

One of the most studied types of MBEAs are the Estimation of Distribution Algorithms (EDAs). EDAs try to find which parts of solutions are important by estimating the distributions of partial solutions. Moreover, EDAs are not very different from traditional EAs in the sense that they only differ in their crossover and mutation operators, which are replaced by ML models. Similarly to EDAs, the techniques proposed in this paper also use ML to create a model of the building blocks that fits the distribution of partial solutions most accurately. However, in this study the model is only used by the crossover operator, this choice was made to have a more focused evaluation. However, for future research, it could be interesting to also investigate the effectiveness of the proposed techniques on the mutation operator. Additional information on MBEAs and their variants can be found in a survey by Cheng et al. [5].

3 APPROACH

In this work, the effectiveness of preserving inter-gene dependency structures is evaluated in a practical setting. To do this we present two novel techniques. These techniques build upon MOSA. While MOSA has been proven to be very effective in generating test-cases, it does not address the disruptiveness of the variational operators. To improve the MOSA algorithm we will extract a building block model from a subset of the current population. This subset consists of the first two Pareto fronts, which are sets of non-dominated solutions i.e. a solution within one front does not dominate any of the other solutions in that front. Finally, the extracted building block model will describe the inter-gene relations and is used during crossover to reduce its disruptiveness. The pseudocode for these operations is presented in Algorithm 1. The two novel techniques are equal in usage and only differ in how they learn the model. It must be noted that the algorithms do not always use crossover. Instead, there is a certain probability of doing crossover.

The first technique will be called Agglomerative Clustering MOSA (ACMOSA). ACMOSA performs clustering based on a similarity matrix that describes the similarities between parts of the genes of individuals in the population. These similarities can indicate dependencies between those genes which means they form a building block.

The second technique is called Gene-pool Optimal Mixing MOSA (GOMOSA) which uses the GOMEA algorithm described by Orphanou et al. to find the structure of a BN that represents the inter-gene relations [10]. The GOMEA algorithm uses Agglomerative Clustering internally to calculate a linkage tree for Optimal Mixing. The most optimal BN is then converted to building blocks.

Algorithm 1: ACMOSA/GOMOSA

input :
$P = \{p_1, ..., p_m\}$ the population
output:
A test suite T
begin
  $t \longleftarrow 0$
  $P_t \longleftarrow$RANDOM-POPULATION(M)
  while *not*(*search_budget_consumed*) **do**
    if t *modulo recalculation_interval* $= 0$ **then**
      model $\longleftarrow$EXTRACT-MODEL(P_t)
    $P_{t+1} \longleftarrow \emptyset$
    while $|P_{t+1}| \leq M$ **do**
      $S_t \longleftarrow$ FILTER(P_t)
      $p_a \longleftarrow$ TOURNAMENT-SELECT(S_t)
      $p_b \longleftarrow$ TOURNAMENT-SELECT(S_t)
      child $\longleftarrow p_a$
      if *random* $<$ *crossover_probability* **then**
        child $\longleftarrow$ MODEL-CROSSOVER(p_a, p_b, model)
      P_{t+1}.append(*child*)
      mutant $\longleftarrow$ MUTATE(*child*)
      P_{t+1}.append(*mutant*)
    $F \longleftarrow$ PREFERENCE-SORTING(P_{t+1})
    $P_{t+1} \longleftarrow$ SELECT-USING-FRONTS(F)
    archive $\longleftarrow$ UPDATE-ARCHIVE(P)
    $t \longleftarrow t + 1$
  $T \longleftarrow$ archive

3.1 Learning Inter-gene Dependency Structures

To learn dependencies between parts of test-cases several algorithms and techniques had to implemented in EvoMaster.

3.1.1 Agglomerative Hierarchical Clustering (AC). The AC algorithm is a clustering algorithm used to learn dependencies between variables. It is for example used by the Linkage Tree Genetic Algorithm to learn the linkage tree from the population [13]. ACMOSA uses AC in a very similar manner. The AC algorithm clusters variables together based on similar value expressions within data. This means that AC calculates a similarity matrix of all variables in terms of value expression and then uses that matrix to cluster genes together which are most similar, i.e. the variable groups with the highest correlation. By repeating the clustering process the algorithm eventually ends up with a single cluster containing all the variables. This final cluster is the root of a Linkage Tree, the leaves of the tree are the single variable clusters. However, the most interesting clusters are the internal nodes of the tree, i.e. clusters with $1 < s < m$ where s is the size of the cluster and m the total amount of variables.

3.1.2 Bayesian Networks (BN). BNs have been proven to give excellent insights into the relationship between variables. In this research, BNs are used by the GOMOSA technique to model the dependencies between parts of test-cases. To learn the structure of the BN, several methods can be applied. GOMOSA uses the Genepool Optimal Mixing Evolutionary Algorithm (GOMEA) to find the BN. As shown by Orphanou et al. GOMEA can be very effective in learning the structure of a BN [10].

3.1.3 Fitting criteria. First of all, it is important to understand that there are 2 different search processes at work when using the GOMOSA technique, the main search process which is the original MOSA implementation, and the model fitting process which is the GOMEA technique. These 2 processes use completely different fitness functions as they optimize different types of problems. The fitness function used by MOSA evaluates a test-case by the number of targets it can cover. However, GOMEA does not search for test-cases, instead it searches for a BN that explains the relations between parts of test-cases. In other words, the BN needs to be evaluated on how well it explains the population of test-cases. To do this evaluation the Bayesian Dirichlet equivalent uniform (BDeu) scoring function will be used, which is shown in Equation 1 [4]. The BDeu scoring function computes the posterior probability of the data given a model. In our case, the data is equal to a population of test-cases.

$$BDeu(B,T) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\left(log\left(\frac{\Gamma(\frac{1}{q_i})}{\Gamma(N_{ij}+\frac{1}{q_i})}\right)+\sum_{k=1}^{r_i} log\left(\frac{\Gamma(N_{ijk}+\frac{1}{r_i q_i})}{\Gamma(\frac{1}{r_i q_i})}\right)\right) \quad (1)$$

where:

- n is the number of variables.
- q_i is the number of parent configurations of variable i.
- r_i is the number of values the i-th variable can be.
- N_{ij} is the number of instances in the data where the parent variables of the i-th variable take their j-th configuration.
- N_{ijk} is the number of instances in the data where the parent variables of the i-th variable take their j-th configuration and the i-th variable takes its k-th value.
- $\Gamma()$ is the gamma function [3].

3.1.4 Repair operator. The structure of a BN represents a Directed Acyclic Graph (DAG) which means that variables cannot have cyclic dependencies in the model. Using GOMEA or other EAs to find the structure does not give any assurance that the resulting graph is acyclic. To solve this each path in the graph is iterated over while keeping track of the visited nodes. If for a certain path an already visited node is found again, then the connection to that node is removed to make the path acyclic. This operation is performed on each new solution before it is evaluated.

3.1.5 Parents. A limitation of the BDeu scoring method is that q_i grows exponentially with the number of parents, e.g. if every variable can take 2 values then for each parent of variable i, q_i doubles. In other words, if variables are allowed to have many parents the BDeu scoring function becomes infeasible to compute. To solve this problem the amount of parents per variable is curbed by randomly removing parents until the given thresh-hold is met. Additionally, curbing the number of parents reduces the maximum complexity of the graphs. For this research, the maximum amount of parents has been set to 2. This value was chosen because EvoMaster seems to mostly generate tests with only one action when running VMOSA. Meaning that actions are not very likely to have many

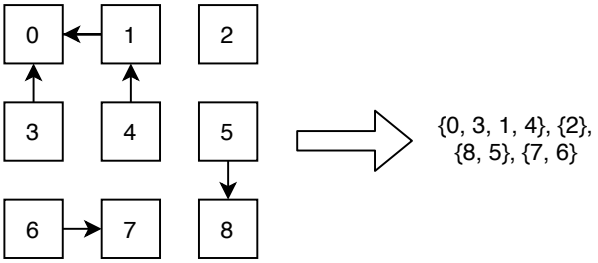

Figure 1: BN to Building Block conversion

dependencies. However, in future research, it would be interesting to explore different values.

3.1.6 Building Blocks. After learning the BN structure from the population it is converted to a building block model that is usable by the crossover operator. The conversion puts variables that depend on each other in the same block, as can be seen in Figure 1. In this case, the numbers in the building blocks represent the indices of actions that are dependent on each other.

3.1.7 Selection. To find a model for the inter-gene dependencies a population of solutions is used. However, the choice of this population has a major impact on the accuracy of the model. Several methods have been tried, e.g. filtering the population based on the length of test-cases or using the elitist archive. Taking the first two Pareto fronts of the current population up being the most effective method. This conclusion is based on the number of targets covered by the algorithm within a certain timeframe.

3.1.8 Overhead. Computing the model turns out to be a very time-consuming process, especially for the GOMOSA technique. Recomputing the model every generation makes GOMOSA perform the worst of all techniques. Recomputing once every so many generations did improve the performance time-wise. However, by doing so the model becomes outdated for several generations making the model less accurate. So there is a trade-off between having an up-to-date model and dedicating the least amount of time to recalculating the model. If the model stays outdated for too many generations it will no longer provide a benefit. If the model is updated every generation a lot of time will be lost which results in lower achieved coverage. This trade-off heavily depends on the size of the API. Smaller APIs can recalculate the model more often since fewer possible actions result in faster model calculation. This is the case because the Agglomerative Clustering Algorithm performed by both ACMOSA and GOMOSA grows quadratically with the number of possible actions.

3.1.9 Encoding. The techniques used to extract a dependency model from data require the data to be of a fixed size. To accomplish fixed-size data the solutions in the population need to be encoded. There are multiple ways to do the encoding, some more difficult than others. For this research, the solutions were encoded by the presence of the possible action genes. This means that for each of the k solutions in the population a vector will be created of size m where m is the total amount of possible actions. If a solution contains the i-th action, with $0 < i < m$, then the i-th position in the vector will be a 1, otherwise a 0. This way we can measure whether certain actions appear together often (positive correlation), or that actions never occur together (negative correlation). While this method of encoding is fairly simple and works well it causes the data to lose the positional information of the actions, i.e. a certain request may appear in front of another request but never behind it. Future research could investigate whether different encoding techniques without information loss benefit the effectiveness of the proposed techniques.

3.2 Building Block based Crossover

One-point crossover can be easily modified to use the building block model. Similar to One-point crossover, the child of the two parents will be a copy of the first parent with parts of the second parent. However, instead of using a random crossover point, it uses the indices of a random building block from the model to crossover certain actions from the second parent. By using this type of crossover the building blocks stay intact, thus reducing the disruptiveness of the crossover operator. The pseudocode for this type of crossover can be found in Algorithm 2.

Algorithm 2: MODEL-CROSSOVER

input :
p_a the first parent solution
p_b the second parent solution
model = $\{b_1, ..., b_n\}$ the model containing building blocks
output:
A child solution c
begin
  $b_r \longleftarrow$ PICK-RANDOM(model)
  $c \longleftarrow$ COPY(p_a)
  for $i \in b_r$ **do**
    $c_i \longleftarrow p_{bi}$

3.2.1 Filtering. After closer inspection of the search process of EvoMaster, it has been noted that in its current state EvoMaster tends to prefer solutions with a small number of actions over solutions with more action genes. This preference results in a population where most solutions contain a single action. This means that doing crossover does not make sense as it would result in copying one of the parents. To overcome this problem the population used for crossover is filtered to make sure it only contains multi-action solutions.

4 EVALUATION

The main goal of this article is to evaluate the effectiveness of using inter-gene dependency structures, also known as building blocks, during crossover in a practical setting such as test case generation. In this section, the evaluations of the novel methods are described. Several plots and tables are included. However, to keep a clear overview within this section all plots are from the NCS benchmark problem, all relevant plots of the other benchmark problems can be found in the Appendix.

Table 1: Benchmark problem size & parameter

Problem	No. Actions	Recalculation interval
ncs	6	4
scs	11	16
features-service	18	16
scout-api	49	64
proxyprint-kitchen	74	64

Table 2: Experimental setup system specifications

Part	Info
CPU	Intel Core i7-7700HQ 2.80GHz
RAM	16GB DDR4 2400MHz
OS	Ubuntu 18.04

To evaluate the effectiveness of ACMOSA and GOMOSA, we tested several benchmark RESTful APIs made available by the EvoMaster benchmark project (EMB) [1]. These APIs are of varying sizes to ensure that the results are not problem-specific. The chosen benchmark APIs are:

- **NCS:** an artificial API
- **SCS:** an artificial API
- **Features-service:** a MicroService for managing products Feature Models
- **Scout-API:** an API created by MIT
- **Proxyprint-kitchen:** a platform for printshops.

The size of the benchmark problem is defined by the number of possible actions the API provides. The number of actions per API can be found in Table 1 together with the recalculation interval that was used for that benchmark problem. The values for the recalculation interval are based on the size of the API. For this evaluation the embedded EvoMaster controllers of these APIs are used, these controllers are supplied by the EMB project. The controllers provide EvoMaster with API documentation from which EvoMaster extracts what HTTP requests can be made. Additionally, the controllers allow EvoMaster to use a white-box approach by providing an interface that retrieves the line and target coverage achieved by certain test-cases, in contrast to the black-box approach where the only feedback is the status code in the response of the API.

Specifications. All of the experiments of this research have been conducted on the same system, namely a Dell Inspiron 15 laptop. Additionally, all the experiments used a local connection to make sure that internet latency does not impact the results. During the experiments, other processes were kept to a minimum. In Table 2 the relevant specifications of the system can be found.

Results. Each algorithm ran 20 times for 300 seconds for each benchmark problem. Table 3 reports the average target coverage achieved by VMOSA, MIO, MOSA, and the novel techniques ACMOSA and GOMOSA. The table also reports the median target coverage and the inter-quartile range (IQR) to give a sense of the spread of the results. To measure the statistical significance of the difference between the results of the VMOSA baseline and the technique under evaluation, the non-parametric Wilcoxon signed-rank test has been used [6]. The p-values resulting from the Wilcoxon signed-rank tests are reported in the table. Significant p-values, i.e. $p < 0.05$ show that the results of the algorithm are unlikely to be sampled from the same distribution. Finally, the Vargha-Delaney A_{12} statistic is provided which describes the magnitude of difference between the VMOSA baseline and the technique under evaluation [14].

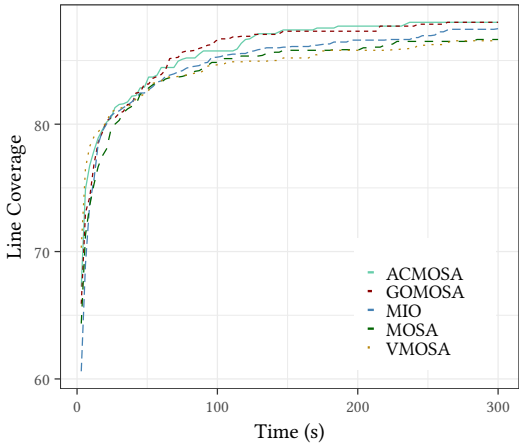

Figure 2: NCS: Average line coverage over time

The Wilcoxon test indicated that ACMOSA (Mdn = 617) and GOMOSA (Mdn = 617) have a significantly higher amount of covered targets than the VMOSA baseline (Mdn = 609.5) for the NCS benchmark, ($p < 0.01$). Both ACMOSA and GOMOSA have a *large* effect size for the NCS problem. Although MIO (Mdn = 614) also has a significantly higher target coverage ($p < 0.02$), ACMOSA and GOMOSA perform slightly better for NCS. Meanwhile, MOSA (Mdn = 611) performed roughly equal to the baseline. For the SCS problem MIO (Mdn = 780.5) is the best performing algorithm while both ACMOSA (731) and GOMOSA (Mdn = 746) are performing significantly worse than the VMOSA baseline (Mdn = 763), ($p < 0.01$). MOSA (Mdn = 709.5) also performed significantly worse than the baseline ($p < 0.01$). The FEATURES benchmark seems to leave little room for improvement as none of the algorithms did significantly better than the baseline (Mdn = 394). ACMOSA (393) and GOMOSA (Mdn = 394) performed mostly equal to the baseline while both MIO (Mdn = 394) and MOSA (Mdn = 395) performed slightly better but with a larger inter-quartile-range. The SCOUT API is another example where ACMOSA (Mdn = 1578) and GOMOSA (Mdn = 1582) perform significantly better than the VMOSA baseline (Mdn = 1544), ($p < 0.01$). Both also perform better than MIO (Mdn = 1496). However, for this benchmark, it seems that crossover in general provides a significant benefit as MOSA (1583) performs roughly equal to ACMOSA and GOMOSA ($p < 0.01$). Finally, for the PROXYPRINT benchmark only MIO (Mdn = 1433) provided a significantly higher coverage than the baseline (Mdn = 1197), ($p < 0.01$). ACMOSA (Mdn = 1262), GOMOSA (Mdn = 1234), and MOSA (1185) all performed roughly equal to the baseline without significant differences in the distributions.

Table 3: Average target coverage after 20 runs of 300 seconds along with the median, the IQR, the p-values resulting from the Wilcoxon test, and the Vargha-Delaney A_{12} value together with the Verbal effect size. All techniques except MIO were compared against the VMOSA baseline without crossover. $A_{12} < 0.5$ means that VMOSA performs better while $A_{12} > 0.5$ means that the algorithm under evaluation performs better. Significant differences compared to the baseline are marked in bold.

NCS	average	standard deviation	median	IQR	p-value	A_{12}	Magnitude
VMOSA (baseline)	605.95	11.33	609.5	11.75	–	–	–
MIO	613.90	5.29	614	7.25	**0.02**	**0.77**	**large**
MOSA	609.95	8.02	611	11.75	0.47	0.60	small
ACMOSA	617.85	2.81	617	3.50	**< 0.01**	**0.92**	**large**
GOMOSA	617.50	2.40	617	3.00	**< 0.01**	**0.91**	**large**
SCS							
VMOSA (baseline)	758.35	21.59	763	21.50	–	–	–
MIO	785.50	33.89	780.5	41.75	**< 0.01**	**0.77**	**large**
MOSA	710.95	33.78	709.5	52.50	**< 0.01**	**0.10**	**large**
ACMOSA	725.45	26.02	731	25.50	**< 0.01**	**0.14**	**large**
GOMOSA	737.30	29.47	746	41.75	**0.02**	**0.28**	**medium**
FEATURES							
VMOSA (baseline)	393.82	3.83	394	2	–	–	–
MIO	409.82	33.09	394	11	0.1693	0.57	negligible
MOSA	396.06	5.31	395	7	0.2552	0.65	small
ACMOSA	393.35	2.79	393	3	0.6775	0.43	negligible
GOMOSA	394.35	2.06	394	3	0.8356	0.51	negligible
SCOUT							
VMOSA (baseline)	1543.66	34.49	1544	37.50	–	–	–
MIO	1506.26	49.26	1496	60.50	0.12	0.31	medium
MOSA	1579.33	26.14	1583	29.00	**< 0.01**	**0.87**	**large**
ACMOSA	1584.53	37.61	1578	20.50	**< 0.01**	**0.92**	**large**
GOMOSA	1585.20	40.28	1582	17.00	**< 0.01**	**0.90**	**large**
PROXYPRINT							
VMOSA (baseline)	1196.95	108.00	1197	164.50	–	–	–
MIO	1431.35	9.90	1433	17.50	**< 0.01**	**1**	**large**
MOSA	1185.30	134.83	1185	127.25	0.64	0.47	negligible
ACMOSA	1222.95	141.44	1262	216.25	0.57	0.56	negligible
GOMOSA	1193.30	139.38	1234	167.75	0.65	0.52	negligible

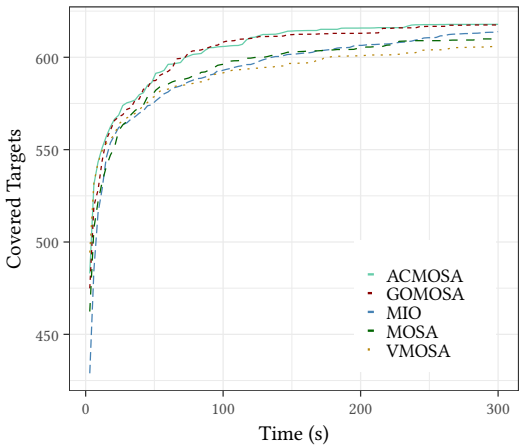

Figure 3: NCS: Average covered targets over time

Table 3 shows that the novel techniques perform differently for each problem, for SCS they perform worse than the other techniques, for FEATURES and PROXYPRINT they perform equally good, and for NCS and SCOUT, ACMOSA and GOMOSA perform significantly better. This discrepancy can be due to multiple factors. For example, the size of the benchmark APIs has a major impact on the speed at which the model can be calculated, which in turn means that the novel techniques waste precious search time reducing the chance of covering new targets. Another, very important factor, maybe even the most important factor is the dependence between parts of the API i.e. there can only be dependencies between parts of test-cases if there are dependencies between parts of the API. If for example an API has been written in such a way that the HTTP request can never influence the outcome of another HTTP request then the novel techniques logically do not provide any benefit. Furthermore, there possibly is a maximum on the number of targets that can be covered per API. This also seems to be the case when looking at any of the figures that show the covered targets over time. However, this maximum is not the actual maximum necessarily as the number of covered lines is not 100% at the end of

the 300 seconds. This becomes clear when comparing Figures 2 and 3. Anyhow, this maximum creates a boundary for all algorithms which makes it more difficult for an algorithm to do significantly better than others.

Another factor is that the algorithms in EvoMaster start their search with a population that already has a high target coverage. This is due to the sampling EvoMaster does based on the API documentation. In other words, the starting population of the algorithms has such a high coverage that there isn't much room for improvement anymore, which again makes it more difficult for an algorithm to do significantly better than other algorithms.

Finally, there is a large difference in the amount of time a test evaluation costs between the benchmark problems. For example, the proxy-print problem takes an average of 250 ms per test-case evaluation, while the NCS problem only has an average of 2 ms per test-case. Of course, this depends on the number of actions within the test-case, however, it seems that PROXYPRINT is a lot slower in handling the requests. Moreover, a more time-consuming test evaluation means that fewer generations can be processed. This becomes apparent when comparing the NCS benchmark to the PROXYPRINT benchmark as the VMOSA baseline can process around 1500 generations for the NCS benchmark while only processing around 40 generations for the PROXYPRINT benchmark. Doing fewer generations coupled with a large overhead due to the size of the PROXYPRINT API could be the cause of ACMOSA and GOMOSA performing less effective than on the NCS problem.

Especially the sampling makes it hard to compare the results of the different algorithms. It would have been fairer to compare the algorithms starting from nothing. The MIO algorithm in particular uses the sampling a lot giving it a significant advantage for certain benchmark problems. However, this paper focuses on the real-world application of these algorithms so the most time-efficient algorithm wins. Anyhow, as the newly proposed algorithms are for most benchmark problems at least equal in terms of efficiency they still are a beneficial contribution to the EvoMaster tool and can in some cases provide a significant speedup.

From the flattening of the curve in Figure 3 it can be speculated that there is a limit on the maximum amount of covered targets. This maximum creates the issue that given sufficient time all algorithms could reach this maximum. However, ACMOSA and GOMOSA reach this maximum earlier in time. To properly compare these results a boxplot of the covered targets after 100 seconds is supplied in Figure 4. At this point, ACMOSA and GOMOSA seem to reach the maximum while the other approaches are not quite there yet. It becomes clear that for the NCS API the model-based crossover can find more targets in a shorter amount of time than the other approaches.

5 RESPONSIBLE RESEARCH

To ensure reproducibility we used the publicly available benchmark APIs of the EvoMaster benchmark project. The implementation of the two proposed algorithms is also made public through a pull request to the original EvoMaster project. Additionally, the modifications to the EvoMaster project have been fully documented and sufficiently tested to minimize the likelihood of erroneous code.

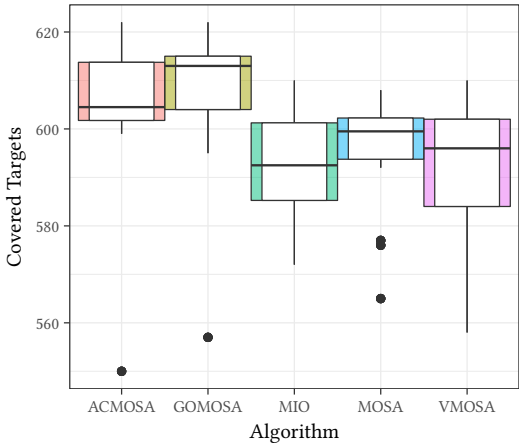

Figure 4: NCS: Boxplot comparison covered targets after 100 seconds

To overcome the randomness of the outcomes of the EAs, we executed each algorithmic variant 20 times for 300 seconds and reported the average performance over time. Additionally, we provided the results of statistical tests as evidence for the significance of the difference between the performances of the algorithms. To ensure a controlled environment that provides a fair evaluation of the algorithms, all experiments have been conducted on the same system and interfering processes were kept to a minimum. Finally, and this goes without saying, the results have not been modified or cherry-picked. -

6 DISCUSSION

In summary, the two proposed techniques, ACMOSA and GOMOSA can for some problems provide significant improvement to the efficiency in covering targets. From the results, it can be deduced that preserving inter-gene relations is beneficial in the search process of generating test-cases in some cases. In this study, ACMOSA and GOMOSA turned out to be particularly beneficial to the NCS benchmark API. They also performed well on the SCOUT benchmark, however, as MOSA performed almost equally well, it cannot be concluded that the inter-gene relations play a huge role. Furthermore, the FEATURES benchmark seems to suffer most from the hypothetical maximum coverage boundary as none of the algorithms performed significantly better than the others. This conclusion can also be made when we look at Figure 7 which shows that this boundary is reached relatively early in the search process i.e. after around 100 seconds while the entire search is 300 seconds. The PROXYPRINT benchmark did not provide a clear winner between the MOSA variants while MIO outperformed them all. However, as mentioned before the PROXYPRINT benchmark has a significantly longer test-case evaluation time than for example the NCS benchmark. The longer evaluation time together with the size of the API curbed the number of generations that could be processed in the 300-second time-frame, so much so that the model calculation only happened once or twice during the search process. The impact of the size of the API on the GOMOSA algorithm is especially visible

in Figure 8 on the PROXYPRINT benchmark. In this figure, it is clear that calculating the model at the beginning of the search takes a lot of time which is a likely explanation as to why GOMOSA starts with the worst performance but catches up after a while. Finally, the SCS problem did not benefit from any type of crossover as MIO and VMOSA were the most effective.

Overhead. As mentioned before computing the Bayesian network model gives GOMOSA a large overhead in time complexity. Improving the complexity in finding this model can potentially boost the effectiveness of the algorithm even more. The same applies to the ACMOSA technique which has less overhead.

Recommendations. The current implementation of both ACMOSA and GOMOSA makes use of the current population to calculate a model from, this population consists of test-cases which cover certain targets. It could be interesting to instead use manually written tests and learn the model from those. By doing so the techniques also lose the overhead as they will not have to recompute the model anymore, i.e. the manually written tests do not change, and recomputing is thus less use-full. Additionally, future research should investigate other less computationally expensive ML methods to improve the benefit of the techniques for large APIs.

For future research, it could also be interesting to evaluate the novel techniques for longer periods of time. This allows investigating the exploratory capabilities of the techniques. That is, are they able to find more coverage than other techniques in general or do they only converge faster? Additionally, it would be interesting to evaluate the techniques on a more diverse set of benchmark problems including more complex and larger APIs.

7 CONCLUSIONS

In this article two novel techniques, ACMOSA and GOMOSA which build upon the MOSA algorithm have been proposed. These techniques use previously found solutions to extract dependency models that explain relations between parts of the solutions. The models are then used during the crossover operation to preserve the inter-gene relations, reducing the disruptiveness of the crossover operation to the search process.

As shown in the result section both of the novel techniques perform differently for each problem, however for some problems the new techniques provide a significant benefit. The difference in performance between the benchmark problems is most likely due to a combination of the size of the problem and the level of dependence between the API endpoints. For the software engineers that want to use EvoMaster, it is thus advised to determine these two properties for the API under test before a certain technique is applied.

We can thus conclude that intelligent evolutionary operators do for certain problems have a significant impact on EvoMaster's performance, and that future research should investigate whether the application of these novel methods is beneficial in other settings. Another topic of interest for future research is to investigate other less time-consuming modeling techniques.

REFERENCES

[1] Andrea Arcuri. 2018. EvoMaster: Evolutionary Multi-context Automated System Test Generation. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 394–397.

[2] Andrea Arcuri. 2019. Many Independent Objective (MIO) Algorithm for Test Suite Generation. *CoRR* abs/1901.01541 (2019). arXiv:1901.01541 http://arxiv.org/abs/1901.01541

[3] Emil Artin. 2015. *The gamma function*. Courier Dover Publications.

[4] Wray Buntine. 1991. Theory Refinement on Bayesian Networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence (UAI'91)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 52–60.

[5] Ran Cheng, Cheng He, Yaochu Jin, and Xin Yao. 2018. Model-based evolutionary algorithms: a short survey. *Complex & Intelligent Systems* 4, 4 (2018), 283–292.

[6] William Jay Conover and William Jay Conover. 1980. Practical nonparametric statistics. (1980).

[7] Laura Dioşan and Mihai Oltean. 2006. Evolving Crossover Operators for Function Optimization. In *Genetic Programming*, Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 97–108.

[8] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[9] Jong-Min Kim, Yoon-Sung Jung, Engin A Sungur, Kap-Hoon Han, Changyi Park, and Insuk Sohn. 2008. A copula method for modeling directional dependence of genes. *BMC bioinformatics* 9, 1 (2008), 225.

[10] Kalia Orphanou, Dirk Thierens, and Peter A. N. Bosman. 2018. Learning Bayesian Network Structures with GOMEA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 1007–1014. https://doi.org/10.1145/3205455.3205502

[11] Annibale Panichella, Fitsum M. Kifetew, and Paolo Tonella. 2015. Reformulating Branch Coverage as a Many-Objective Optimization Problem. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. 1–10.

[12] Jim Smith and T. C. Fogarty. 1996. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computation*. 318–323.

[13] Dirk Thierens. 2010. The Linkage Tree Genetic Algorithm. In *Parallel Problem Solving from Nature, PPSN XI*, Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 264–273.

[14] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the "CL" Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. http://www.jstor.org/stable/1165329

A APPENDIX

A.1 SCS

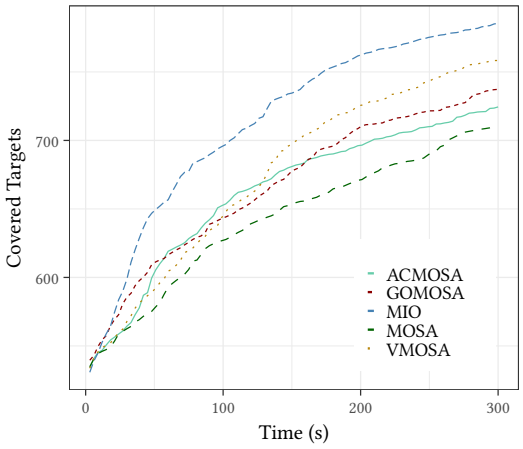

Figure 5: SCS: Average covered targets over time

A.2 Features service

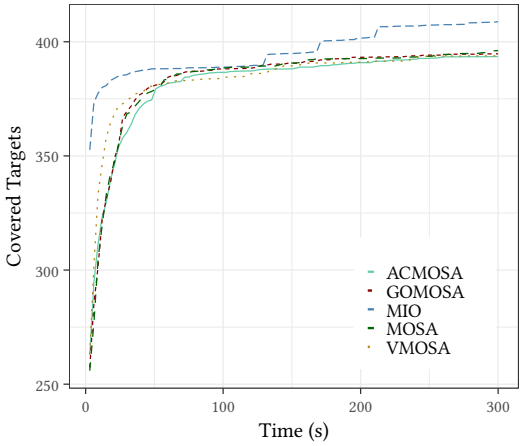

Figure 6: FEATURES: Average covered targets over time

A.3 Scout API

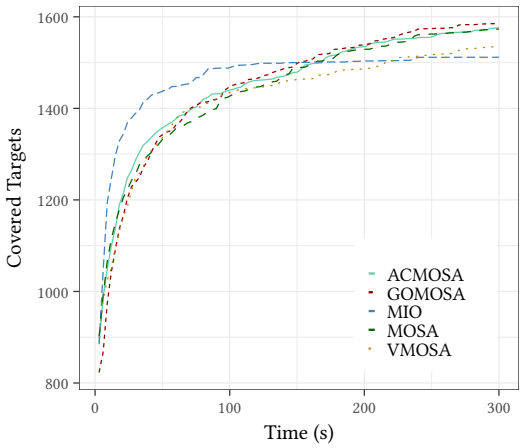

Figure 7: SCOUT: Average covered targets over time

A.4 Proxyprint kitchen

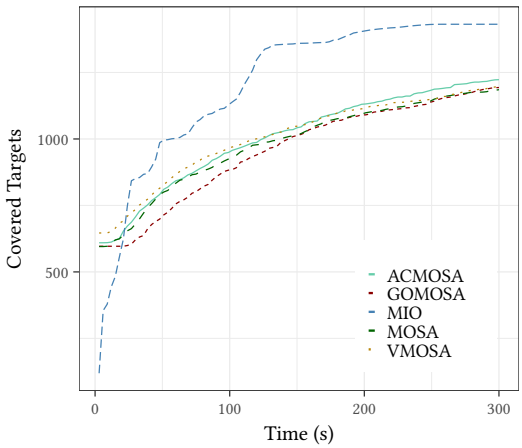

Figure 8: PROXY: Average covered targets over time