Measuring and Maintaining Population Diversity in Search-based Unit Test Generation

Nasser Albunian[1], Gordon Fraser[2], and Dirk Sudholt[1]

[1] The University of Sheffield, UK
[2] University of Passau, Germany

Abstract. Genetic algorithms (GAs) have been demonstrated to be effective at generating unit tests. However, GAs often suffer from a loss of population diversity, which causes the search to prematurely converge, thus negatively affecting the resulting code coverage. One way to prevent premature convergence is to maintain and increase population diversity. Although the impact of population diversity on the performance of GAs is well-studied in the literature, little attention has been given to population diversity in unit test generation. We study how maintaining population diversity influences the Many-Objective Sorting Algorithm (MOSA), a state-of-the-art evolutionary search algorithm for generating unit tests. We define three diversity measures based on fitness entropy, test executions (phenotypic diversity), and Java statements (genotypic diversity). To improve diversity, we apply common methods that fall into two groups: niching (such as fitness sharing and clearing) and non-niching (such as diverse initial populations). Our results suggest that increasing diversity does not have a beneficial effect on coverage in general, but it may improve coverage once the search stagnates.

Keywords: Search-Based Test Generation; Population Diversity

1 Introduction

As software testing is a laborious and error-prone task, automation is desirable. Genetic Algorithms (GAs) are frequently employed to generate tests, especially in the context of unit testing object oriented software. However, a common general issue when applying GAs is premature convergence: If the individuals of the search population all become very similar and lack diversity [6, 25, 26], then the search may converge on a local optimum of the objective function. This reduces the effectiveness of the GA, and in the case of search-based test generation, premature convergence would imply a reduced code coverage.

To avoid such premature convergence, it is important to maintain diversity in the population. Different techniques have been proposed to achieve this at the genotype and the phenotype levels [6, 26]. For example, diversity can be achieved by scaling an individual's fitness based on the density of its niche or by eliminating duplicate individuals from the population. While diversity maintenance has been extensively investigated within different domains of evolutionary algorithms (e.g., [6]), much less is known about diversity in search-based unit test generation.

We empirically investigate the impact of population diversity on the generation of unit tests for Java programs. More specifically, we aim to see whether increasing population diversity leads to a better GA performance, i.e., generating unit tests that achieve higher code coverage. We first adapt common diversity measurements based on phenotypic and genotypic representation to the search space of unit test cases (Section 3). We then study the effects of different diversity maintenance techniques (Section 4) on population diversity and code coverage (Section 5).

2 Search-Based Unit Test Generation

In the context of generating tests for object-oriented programs, a solution is represented as a test case τ which consists of a sequence of calls $\tau = \langle s_1, s_2, \ldots, s_n \rangle$ on the class under test (CUT) [9]. That is, each s_j is an invocation of a constructor of the CUT, a method call on an instance of the CUT, a call on a dependency class in order to generate or modify dependency objects, or it defines a primitive value (e.g., number, string, etc.) As the ideal test case size is not known a priori, the number of statements in a test case is variable and can be changed by the search operators. Consequently, crossover and mutation not only modify the individual statements of a sequence, but can also remove or insert statements.

The fitness functions commonly used are based on code coverage [21]. We focus on branch coverage as it is one of the most common coverage criteria in practice [9], and fitness functions for other criteria are typically based on branch coverage fitness calculations. In the many-objective representation of the unit test generation problem [17], each branch in the CUT forms a single objective. The fitness function of a test τ on branch b_i is $f(\tau, b_i) = al(b_i, \tau) \ + \ \alpha(bd(b_i, \tau))$ [13], where bd is the branch distance, α is a normalization function mapping the branch distance to the range $[0, 1]$, e.g. $x/(x + 1)$, and al is the approach level, the distance between the closest control dependency of the target node executed by a test and the target node in the control dependency graph.

A common approach lies in evolving sets of unit tests [9] using single-objective optimization, or individual test cases using many-objective optimization [17]. The Many-Objective Sorting Algorithm (MOSA) has been shown to generally perform best [3]. MOSA (Algorithm 1) starts with an initial population of randomly generated test cases (line 4), and applies standard genetic operators (line 7). To generate the next generation, parents and offspring are combined (line 8) and sorted using the preference criterion and non-dominance relation (line 9). The preference criterion identifies a subset of test cases that have the lowest fitness values for uncovered branches, which are assigned rank 0, while the traditional non-dominated sorting used by NSGA-II is applied to rank the remaining test cases into further fronts. Selection is then applied based on the assigned ranks starting at the first front, until reaching the population size n (lines 12-16). When the number of selected test cases exceeds the population size n, the individuals of the current front F_r are sorted based on the crowding distance (line 17) and only those individuals with higher distance are selected. At the end of each generation, MOSA updates an archive with test cases that cover uncovered branches with the lowest possible length (line 19).

Algorithm 1: Many-Objective Sorting Algorithm (MOSA)

```
1  Input: Population size n, Stopping criterion C
2  Output: An archive of best test cases T
3  t ← 0 ;                                              ▷ current iteration
4  P_t ← GenerateRandomPopulation(n)
5  T ← UpdateArchive(P_t)
6  while ¬C do
7  |  P_o ← GenerateOffspring(P_t)
8  |  P_u ← P_t ∪ P_o
9  |  F ← PreferenceSorting(P_u)
10 |  r ← 0
11 |  P_{t+1} ← {}
12 |  while |P_{t+1}| + |F_r| ≤ n do
13 |  |  AssignCrowdingDistance(F_r)
14 |  |  P_{t+1} ← P_{t+1} ∪ F_r
15 |  |  r ← r + 1
16 |  end
17 |  CrowdingDistanceSort(F_r)
18 |  P_{t+1} ← P_{t+1} ∪ F_r ;                         ▷ size n − P_{t+1}
19 |  T ← UpdateArchive(T, P_t)
20 |  t ← t + 1
21 end
22 return T
```

3 Measuring Population Diversity

Population diversity refers to the variety in a population based on the differences at the genotype (i.e., structural) or phenotype (i.e., behavioural) levels. The genotypic diversity measures the structural (i.e., syntactic) differences among the individuals of a population. In contrast, the phenotypic diversity is based on the behavioural (i.e., semantic) differences in the population's individuals.

However, these levels differ among different domains [6], e.g., the structure of an individual in the case of genetic programming (GP) is not similar to the one with other evolutionary algorithms (EAs). For example, McPhee and Hopper [14] considered the number of different nodes in a GP as structural difference among the individuals. In the case of phenotypic diversity, the fitness of the population's individuals can be mainly used to measure the behavioural differences among the individuals where the diversity rate is based on the spread of fitness values [10]. A well-known fitness-based measure is the entropy measure, first proposed by Rosca [23]. The entropy represents the amount of disorder of a population, where an increase in entropy leads to an increase in diversity in the population. Rosca defines diversity based on the entropy as $E(P) = -\sum_k p_k \cdot \log p_k$, where the population P is partitioned according to the fitness value which will result in a proportion of the population p_k that is occupied by the partition k.

In order to determine the influence of the diversity of populations of test cases, a prerequisite is to measure diversity. For this, we adapted three techniques based

Table 1: Two examples (cases) of two test cases (TC_1 and TC_2) that vary in how often they execute the predicates (p_j) of the CUT.

Case 1	p_1	p_2	p_3	p_4
TC_1	2	1	0	3
TC_2	3	4	2	1

Case 2	p_1	p_2	p_3	p_4
TC_1	1	1	0	2
TC_2	1	2	0	1

on the phenotypic and genotypic levels: We measure the phenotypic diversity based on the fitness entropy and test execution traces, and we define a genotypic measurement based on the syntactic representation of test cases.

3.1 Fitness Entropy

The entropy measure adapts the aforementioned principle of fitness entropy. It constructs *buckets* that correspond to the proportions of population that are partitioned based on the fitness values of test cases τ_s in the population μ as:

$$\text{Bucket}(f) \leftarrow |\left\{\tau_i \mid \text{fitness}(\tau_i) = f\right\}| \tag{1}$$

where f is the fitness value that partitioning is based on and τ_i is each individual in the population whose fitness value equals to f. In this case, each bucket of fitness holds the number of individuals that are in the same fitness interval (e.g., the interval of fitness values that are similar in the first five decimal points). The entropy is then calculated based on each bucket of fitness as:

$$\text{Entropy} = \sum_{i=1}^{B} \frac{\text{Bucket}_i}{\mu} \cdot \log\left(\frac{\text{Bucket}_i}{\mu}\right) \tag{2}$$

where B is the number of buckets. However, in a multiobjective context, the fitness entropy is applied on a set of buckets that is constructed for each objective, and then all entropies are added up to calculate the overall entropy.

3.2 Predicate Diversity

As the fitness value in test case generation is mainly based on the branch distance (and other similar measurements), there is the potential issue that fitness entropy is dominated by a few statements that achieve the best coverage. For example, the fitness value considers only the minimal branch distance for each branch, but ignores all other executions of the same branch. Therefore, we define an alternative phenotypic diversity measurement that takes more execution details into account. The idea behind this measure is to quantify the diversity of the individuals based on an execution profile of the conditional statements in the class under test. To illustrate this, assume two individuals (TC_1 and TC_2) and a class under test (CUT) that has four conditional statements (p_j). Each individual test case covers each predicate in the CUT as shown in Table 1.

The diversity in this case is measured based on how often each predicate is executed by each individual, e.g., p_3 is covered 2 times by TC_2, while it is not

```
1 @Test public void test1() {
2   String string0 = "foo";
3   String string1 = "bar";
4   Foo foo0 = new Foo(string0, string1);
5   String string2 = foo0.bar();
6 }
```

```
1 @Test public void test2() {
2   Bar bar0 = new Bar();
3   String string0 = "bar";
4   Foo foo0 = new Foo(bar0, string0);
5   String string1 = foo0.bar();
6   Foo foo1 = new Foo(string1, string0);
7   String string2 = foo1.bar();
8 }
```

Fig. 1: Two automatically generated example test cases to illustrate statement difference.

covered by TC_1 as shown in Case 1. Predicate diversity is calculated by counting the number of times each predicate p_j in CUT is covered by each individual TC_i, resulting in vectors V_1 and V_2. The distance between TC_1 and TC_2 is calculated using the Euclidean distance between V_1 and V_2, which is also calculated for each pair of individuals in the population. The use of Euclidean distance as a population diversity measure is shown to be effective in measuring the behavioural diversity, and controlling the evolution process [1, 20]. The diversity of the two cases shown in the example are 4.243 for Case 1 and 1.414 for Case 2, which means that the first two test cases are more different than the second two. Therefore, the overall population diversity is the average of all pairwise distances between all individuals, and is calculated as follows:

$$\text{diversity}(P) = \frac{\sum_{i=0}^{|P|} \sum_{j=0, j\neq i}^{|P|} \text{dist}(T_i, T_j)}{|P|\,(|P| - 1)} \tag{3}$$

where P is the population of individual test cases and dist is the Euclidean distance between a pair of test cases.

3.3 Statement Diversity

Genotypic diversity aims to measure the structural differences among the individuals of a population. In our case, the genotypes are the sequences of statements. We measure syntactic difference based on the profile of statements, with normalised variable names. This is important since identical statements at different positions of tests will have different variable names. For example, consider the two test cases in Figure 1: Line 5 in `test1` and Lines 5 and 7 in `test2` are the same except for variable names. To normalise a statement, all variable names are replaced with a placeholder.

To calculate the distance between two test cases `test1` and `test2`, we determine the set of normalised statements contained in both test cases, and then create two vectors representing the number of occurrences of each statement (i.e., for the tests in the example: $V_1 = (2, 2, 1, 1, 0, 0)$ and $V_2 = (1, 1, 1, 2, 1, 2)$, where the 2's in V_2 result from Lines 5 and 7 being counted as the same statement). The distance between two test cases is calculated as Euclidean distance between these two vectors (i.e. 2.82), and the overall diversity is the average distance between all pairs of test cases in the population that is calculated similar to Equation 3.

4 Maintaining Population Diversity

Techniques for maintaining population diversity are typically classified as niching and non-niching techniques [6,25]. Niching techniques try to divide the population into subpopulations to locate multiple optimal solutions. Non-niching techniques maintain diversity in other ways, for example, by increasing the population size, changing the selection pressure, or applying replacement restrictions.

Fitness Sharing (FS) is the most popular niching technique [24] and was proven to be effective in pseudo-Boolean optimisation [15]. It aims to find multiple peaks in the solution space by defining niches around peaks where individuals share the same resource (i.e., fitness value). The idea is to decrease the value of the resource that is shared by the individuals of a niche when the number of individuals is high, and increase it when there are few individuals in a niche, which gives these individuals higher probability to be selected for next generations. The shared fitness of each individual is

$$f_i^{'} = \frac{f_i}{m_i} \quad \text{with} \quad m_i = \sum_{j=1}^{\mu} sh(d_{ij}) \tag{4}$$

where m_i is the niche count, which is defined by measuring the distance among the individuals: here μ is the population size and d_{ij} is the distance between individual i and individual j (e.g., Euclidean distance [24]). The function sh measures the distance between each two individuals in the population as follows:

$$sh(d_{ij}) = \begin{cases} 1 - (d_{ij}/\sigma_s)^{\alpha}, & \text{if } d < \sigma_s \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where σ_s is the peak radius (i.e., sharing radius) and α is the parameter that regulates the form of the sharing function, commonly equal to 1.

Since the fitness function is minimised in our case, the division in Equation (4) is replaced with a multiplication, i.e., $f_i^{'} = f_i \cdot m_i$. This will maximise the shared fitness of the individuals that are dominant in the population and make them less attractive for selection, encouraging other, dissimilar individuals to be selected.

The niche count can be based on any type of distance measurement. In the basic version, the distance is defined as difference between fitness values; for predicate diversity the distance between the predicate execution vectors is used to determine niches, and for statement diversity the distance in statement counts.

Clearing (CL): Clearing is similar to fitness sharing, except that it shares the available resources among the best individuals of each subpopulation rather than all individuals of a subpopulation. That is, it keeps the fitness of the best individuals (i.e., *dominants*) in each subpopulation as they are, and the fitness of the other individuals is cleared (e.g., set to zero). This technique is found to be promising for solving challenging multimodal functions [5] and, in addition, outperforms fitness sharing in dealing with genetic drift [19]. Since we are minimising and the optimal fitness is zero, we set the fitness of cleared individual to a higher fitness value other than zero (e.g., `Integer.MAX_INT`). In order to define a niche, we use the distance between the predicate execution

vectors for predicate diversity, and the distance in statement counts for statement diversity. Both fitness sharing and clearing are calculated for each objective, and when the niche count is based on fitness (i.e., fitness-based sharing), the distance is computed based on fitness values of each objective. However, the two techniques are applied in two steps of Algorithm 1: They are applied on the initial population (after line 4), and on the union of parents and offspring population (after line 8).

Diverse Initial Population (DIP): The initial population is known to have an impact on convergence [11, 27], and its diversity can potentially enhance the performance of the GA [7]. The initial population is diversified by generating a population of random individuals with a size m larger than the intended population size n, and then selecting the most distant n individuals from the population of m individuals based on a diversity measure [28]. We modify Line 4 in Algorithm 1 to generate random individuals of size m and then select only the most n distant individuals to form P_t.

Adaptive Crossover and Mutation Rates (AR): Adaptively changing mutation and crossover rates [6] based on the diversity level is thought to help avoiding convergence to a local optimum. The crossover probability is increased when diversity is high to allow for more exploitation, whereas the mutation probability is increased when diversity is low to allow for more exploration [12]. The crossover probability is adapted as:

$$P_c = \left[\left(\frac{PD}{PD_{\max}} \cdot (K_2 - K_1) \right) + K_1 \right] \tag{6}$$

where K_2 and K_1 define the range of P_c, PD is the current diversity level, and $PD_{\max}$ is the possible maximum diversity level. The mutation probability is adjusted using the following equation:

$$P_m = \frac{PD_{\max} - PD}{PD_{\max}} \cdot K, \tag{7}$$

where K is an upper bound on P_m. Since a variable size representation tends to have multiple different mutation types, i.e., adding, changing, removing statements, the probability of these three operations is adapted by increasing by a random value when the mutation probability increases, and vice versa.

Duplicate Elimination (DE): The purpose of this technique is to remove the similarity between individuals of the population, which has been shown to enhance population diversity and GA performance [4, 22]. Two individuals are considered similar when their distance to each other is zero. To ensure enough diversity in the population, the eliminated individual is replaced with a new generated individual. However, the two similar individuals are evaluated, and the one with the best fitness is kept. This technique is applied after generating the offspring, and more specifically on the union of parents and offspring (after line 8 in Algorithm 1).

Diversity-based Ranking (DR): In the ranking assignment, test cases are selected to form the first non-dominated front based on their objective values

such that a test case x is preferred over a test case y if $f_i(x) < f_i(y)$ where $f_i(x)$ denotes the objective score of test case x_i for branch b_i. When two test cases result in a similar lowest fitness value for a given branch b_i, one of them is chosen randomly to be included in the first non-dominated front. Instead of the random selection, we modify the selection to be based on the diversity such that the test case with high distance from other individuals in the population is preferred to be selected. The distance can be based on the predicate execution vectors (predicate-based) or the statement counts (statement-based).

Diversity-based Selection (DS): Once a rank is assigned to all candidate test cases, the crowding distance is used to make a decision about which test case to select. The basic idea behind the crowing distance is to compute the Euclidean distance between each pair of individuals in a front based on their objective value. In this case, the test cases having a higher distance from the rest of the population are given higher probability of being selected. To investigate the influence of distance-based measures on the selection, we replace the crowding distance with our two measures (i.e., statement-based and predicate-based measures) to calculate the distance between any pair of individuals in each front.

All the previously mentioned techniques can be generalised to any form of Evolutionary Algorithms (EAs), except the last two techniques (i.e., DR and DS) that are designed specifically for MOSA.

5 Empirical Study

The goal of our study is to investigate the evolution of unit tests, and whether maintaining the population diversity during the search has an influence on the performance of GAs. We therefore aim to answer the following research questions:

RQ1: How does population diversity change throughout evolution in MOSA?

RQ2: How effective are diversity maintenance techniques in MOSA?

RQ3: What are the effects of increasing population diversity in MOSA?

For experiments we use EvoSuite [9], which generates JUnit test suites for a given Java CUT and target coverage criterion using different evolutionary algorithms, with MOSA being the most effective algorithm for JUnit test generation [3]. Since open source Java code contains many classes that are either trivially easy to cover, or impossible to cover by EvoSuite, we used the selection of 346 complex classes from the DynaMOSA study [18].

To better understand the influence of the population diversity on the generation of JUnit tests, we conducted an experiment that involves (i) applying each of the three diversity measures defined in Section 3 on each CUT to measure the diversity level throughout the evolution, (ii) applying each of the diversity maintaining techniques defined in Section 4 on each CUT to promote the diversity throughout the evolution. Each of diversity maintaining techniques is integrated into MOSA where its performance compared to the performance of the default MOSA, i.e., without using diversity techniques. Note that each of the diversity techniques is run separately, and therefore the total number of runs is 8 (i.e., a single MOSA run with each of the 7 techniques and one run of default MOSA).

We used the following values for the parameters of the diversity techniques based on preliminary experiments: The sharing radius $\sigma_s = 0.1$; the number of

dominants for clearing is set to 1; the parameters of the diverse initial population are $m = 80$ and $n = 50$; the parameters of adaptive mutation rate are $K_1 = 0.6$ and $K_2 = 0.8$ and the adaptive crossover rate was set to $K = 0.8$.

5.1 RQ1 — How does population diversity change throughout evolution in MOSA?

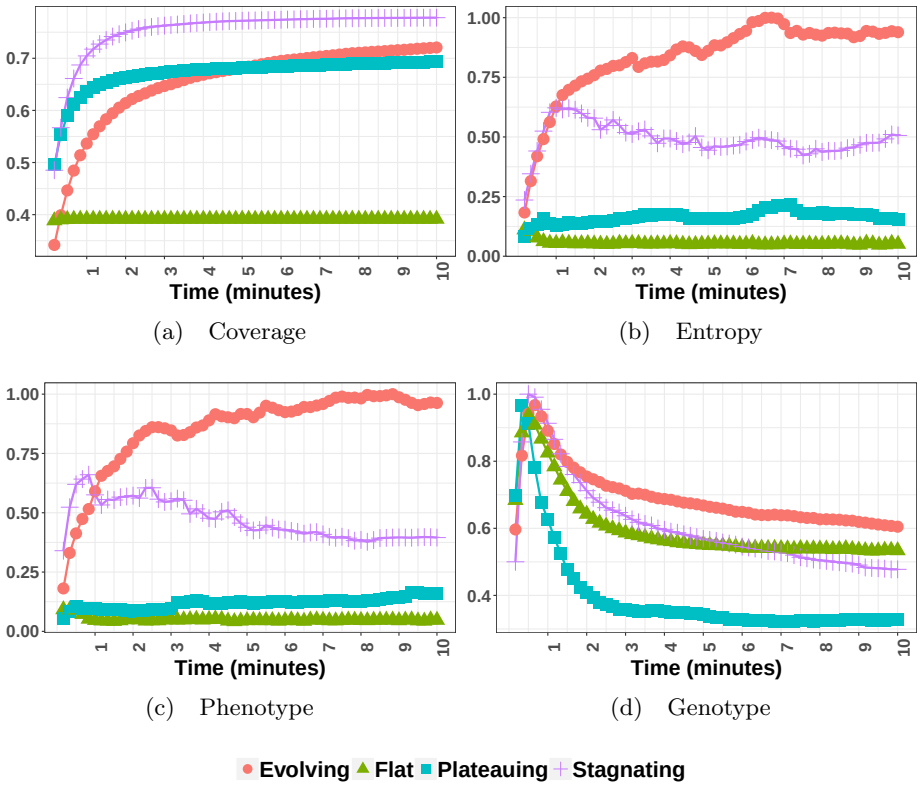
A first step towards understanding the influence of population diversity on the evolution is to measure how the diversity is changed throughout the evolution. For that, we measure the diversity in MOSA to get an idea of whether MOSA is able to maintain a high level of diversity during the search. However, as different CUTs result in different patterns of coverage during the evolution, it is necessary to look at the diversity based on the different coverage patterns.

Past experiments with EvoSuite considered search durations of 1-2 minutes; in order to study the effects on convergence we use a substantially larger search budget. We first performed runs of 30 minutes to observe the development of coverage (the best coverage achieved in the current population) over a long period of time and we used this data to classify CUTs into four disjoint groups as follows:

- **Evolving** contains CUTs where the coverage after 30 minutes is higher (by more than 0.01) than after 10 minutes (93 CUTs).
- **Flat** is the set of CUTs where the coverage never changes (60 CUTs).
- **Stagnating** contains CUTs for which the coverage stagnates after 10 minutes: it is higher than after 2 minutes, but increases by less than 0.01 from 10 minutes to 30 minutes (43 CUTs).
- **Plateauing** contains the remaining CUTs for which the coverage after two minutes is constant (115 CUTs).

We then started 30 runs for each CUT with a search budget of 10 minutes and applied our proposed measures defined in Section 3 to measure the diversity level for each of the four groups, as shown in Figure 2. It is obvious that the diversity behaviour is different among the four groups, and the difference can be seen with the three diversity measures. For the *evolving* group, coverage keeps growing throughout the entire 10 minutes, and this group also shows a continuous growth of entropy and phenotype diversity. In terms of genotype diversity, there is a reduction after an initial sharp growth phase, but less than in all other groups. For the *flat* group, the phenotype diversity is overall lowest; this is because only very few predicates are covered in the first place, as shown in the coverage plot, and the low entropy. For the *stagnating* group, once the coverage increase slows down, all three diversity measurements go down as well. For the *plateauing* group, once the search converges all diversity measurements drop sharply, and notably the genotypic diversity is lowest of all groups. Overall it seems that, as long as coverage grows, MOSA does well at maintaining diversity. Once coverage stagnates, the population loses diversity. To some extent, this can be explained by EvoSuite's ranking mechanism: If two individuals have the same fitness value, then the shorter of the two is preferred; this is also used in MOSA's rank-based preference sorting. Shorter individuals by construction will have less diversity.

***RQ1**: MOSA maintains high diversity while coverage increases, but diversity drops once a maximum coverage has been reached.*



(a) Coverage (b) Entropy

(c) Phenotype (d) Genotype

Fig. 2: Average values for coverage and population diversity throughout evolution in MOSA for the four groups of CUTs.

5.2 RQ2 — How effective are diversity maintenance techniques in MOSA?

In RQ1 we saw a general tendency that diversity drops once the coverage stops increasing. We therefore would like to see which diversity maintenance techniques succeed at increasing the population diversity during evolution. For that, we apply the techniques mentioned in Section 4 on all classes.

Table 2 summarises the results of the average diversity among the four groups with each technique based on different distance measures (i.e., predicate, statement, and fitness). We can clearly see that the three diversity measures indicate that many of the diversity maintenance techniques are able to promote diversity higher than MOSA. The entropy measure indicates that diversity is the lowest with MOSA compared to all other techniques although the difference is negligible when compared to the statement-based Diversity-based Selection (DS), and fitness-based fitness sharing increases entropy the most. A similar trend can be observed for phenotypic diversity, where fitness-based fitness sharing results

Table 2: Average diversity over time when applying diversity maintenance techniques based on different distance measures

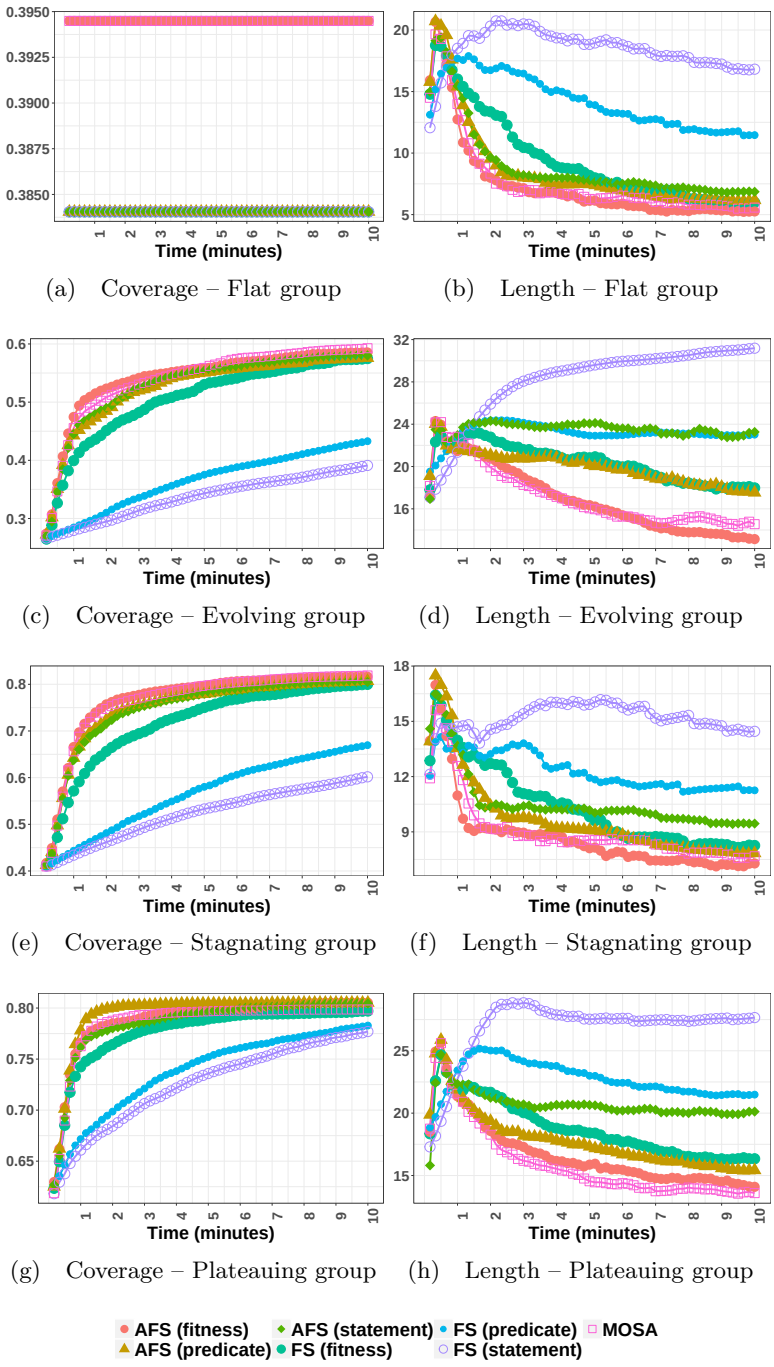
Technique	Phenotype	Genotype	Entropy
MOSA	0.7206	0.6825	0.6761
AR (predicate)	0.5719	0.7854	0.9126
AR (statement)	0.6708	0.7702	0.7005
CL (predicate)	0.8740	0.9470	0.8769
CL (statement)	0.8388	0.9075	0.8503
DE (predicate)	0.7696	0.9350	0.8675
DE (statement)	0.7020	0.9051	0.8095
DIP (predicate)	0.7675	0.6593	0.6976
DIP (statement)	0.7179	0.6851	0.7084
DR (predicate)	0.7996	0.7749	0.7387
DR (statement)	0.6142	0.7775	0.6798
DS (predicate)	0.8737	0.9331	0.7422
DS (statement)	0.8368	0.9053	0.8543
FS (fitness)	0.8893	0.7402	0.9534
FS (predicate)	0.8877	0.9022	0.9353
FS (statement)	0.8881	0.9364	0.8836

in the overall highest diversity. In contrast to entropy, not all techniques succeed in increasing phenotypic diversity; for example, predicate-based Adaptive Rates (AR) results in the lowest diversity. The genotype measure indicates a wide range of effectiveness, with Clearing (CL) based on predicate distance and the statement-based fitness sharing as the most successful in promoting genotypic diversity. However, several other techniques lead to a reduction on genotypic diversity, surprisingly in particular Diverse Initial Population (DIP).

***RQ2**: Most diversity maintenance techniques succeed at increasing diversity, but there are exceptions. Fitness sharing achieves the most consistent increase.*

5.3 RQ3 — What are the effects of increasing population diversity in MOSA?

To investigate the impact of diversity on the performance of MOSA, we look at the achieved coverage and the average size of the individuals in the population throughout the evolution. We focus on fitness sharing (using all three diversity metrics), since RQ2 suggested that this is the most effective technique to increase diversity. In addition to the naive application of fitness sharing at all times, we also consider an *adaptive* version, where we apply fitness sharing only when diversity drops below a certain threshold, and once the diversity level exceeds the threshold, the diversity technique is not applied. We empirically determined a threshold of 60%. Figure 3 shows the results of the best coverage in the population and the average length of all test cases in the population for MOSA with and without fitness sharing (FS (fitness/predicate/statement)) during the evolution.



(a) Coverage – Flat group (b) Length – Flat group

(c) Coverage – Evolving group (d) Length – Evolving group

(e) Coverage – Stagnating group (f) Length – Stagnating group

(g) Coverage – Plateauing group (h) Length – Plateauing group

Fig. 3: Coverage and length over time with MOSA, fitness sharing (FS), and adaptive fitness sharing (AFS) per four groups of CUTs

Table 3: Number of classes where adaptive fitness sharing has an increased/decreased/equal coverage compared to MOSA, the average effect size $\hat{A}_{12}$ and the number of classes for which this comparison is statistically significant ($\alpha = 0.05$).

Technique	increased coverage			decreased coverage			equal
	#classes	#sig.	$\hat{A}_{12}$	#classes	#sig.	$\hat{A}_{12}$	#classes
Fitness	99	6	0.52	87	27	0.48	125
Predicate	52	2	0.51	141	39	0.48	118
Statement	77	4	0.51	116	33	0.49	118

For the *flat* group, there is a small difference in coverage as MOSA and fitness-based fitness sharing result in slighlty higher coverage than the other techniques. The length plot shows how MOSA removes all redundancy from the population, while adding diversity leads to larger individuals. In particular, genotype-based fitness sharing has quite dramatic effects on size. Fitness-based sharing (adaptive and non-adaptive) has the smallest effects on size. For the *evolving* group, MOSA achieves the highest coverage, while maintaining a somewhat constant population size. Generally, fitness sharing slightly increases size, quite dramatically so for genotype-based fitness sharing. Adaptive fitness-based sharing even leads to smaller individuals than MOSA. Non-adaptive fitness sharing using genotype and phenotype diversity leads to a notably lower coverage. For the *stagnating* group, the non-adaptive fitness sharing using genotype and phenotype diversity again lead to a notably lower coverage and larger size. This time, however, adaptive fitness-based sharing consistently leads to a higher average coverage than MOSA, and even smaller individuals than MOSA. The *plateauing* group shows similar results to the *stagnating* group, with larger coverage improvement of adaptive phenotype-based sharing. While genotype and phenotype based non-adaptive sharing again lead to lower coverage initially, in this group the size remains large but constant, and the coverage catches up and even overtakes MOSA in the end.

Adaptive fitness sharing leads to higher coverage when the search in MOSA stagnates (e.g., when coverage does not increase). To see whether adaptive fitness sharing is always beneficial, Table 3 shows the number of classes where it increases, decreases, or results in equal coverage with MOSA. Adaptive fitness-based sharing (AFS-fitness) increases coverage on 99 classes and decreases it on 87 classes, albeit having only 6 significant increases as opposed to 27 significant decreases. For adaptive sharing based on predicate (AFS-predicate) and statement (AFS-statement) differences we found more decreases than increases.

The negative effect of diversity on coverage can be explained by the increase in length: Execution of longer tests takes more time, thus slowing down the evolution. Within the time limit of 10 minutes, MOSA executed 629 generations on average. Always applying fitness sharing decreases the average number of generations (457, 344, and 339 generations with FS-fitness, FS-predicate, and FS-statement respectively), but these figures improve when using the adaptive approach (631, 538, and 519 generations with AFS-fitness, AFS-predicate and

AFS-statement, respectively). Consequently, applying adaptive fitness sharing *can* be beneficial, but the question of when it does so is more subtle. Identifying the context in which it can help will be the focus of future research.

***RQ3**: Promoting diversity generally leads to larger tests and reduces coverage. However, when coverage stops growing, adding diversity may improve MOSA.*

6 Related Work

There have been several studies that considered diversity when generating test cases [8, 16]; however, the aim is to increase the diversity of the tests within the final test suite, rather than the individuals in the search population. More recently, Vogel et al. [28] studied the population diversity when evolving tests for mobile applications where they used a distance-based genotypic measure to quantify the diversity of the population. Applying four diversity maintenance techniques on a multi-objective algorithm shows that diversity does not have an effect on coverage, but it finds more faults. However, the negative impact of increasing diversity on the length is also noticed in previous studies [2, 28].

7 Conclusions

Measuring the diversity of generated unit test cases based on entropy, genotypic, and phenotypic levels suggest that diversity maintenance techniques are very effective at promoting diversity throughout the evolution. Looking at their effect on the performance of MOSA, we see that increasing diversity leads to reduced coverage, and a possible increase in the length. However, preliminary results of the adaptive approach suggest that adaptive fitness sharing with a fitness distance metric has a positive impact on coverage on CUTs on which standard MOSA tends to stagnate or plateau during early stages of the evolution. Further work is required to better understand the reasons behind this effect.

References

1. Adra, S.F., Fleming, P.J.: Diversity management in evolutionary many-objective optimization. IEEE Trans. Evol. Comput. 15(2), 183–195 (2010)
2. Albunian, N.M.: Diversity in search-based unit test suite generation. In: Proc. of SSBSE (2017). pp. 183–189. Springer (2017)
3. Campos, J., Ge, Y., Albunian, N., Fraser, G., Eler, M., Arcuri, A.: An empirical evaluation of evolutionary algorithms for unit test suite generation. Inf. Softw. Technol. 104, 207–235 (2018)
4. Chaiyaratana, N., Piroonratana, T., Sangkawelert, N.: Effects of diversity control in single-objective and multi-objective genetic algorithms. J. Heuristics 13(1), 1–34 (2007)
5. Covantes Osuna, E., Sudholt, D.: On the runtime analysis of the clearing diversity-preserving mechanism. Evol. Comput. 27, 403–433 (2019)
6. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM computing surveys (CSUR) 45(3), 1–33 (2013)
7. Diaz-Gomez, P.A., Hougen, D.F.: Empirical Study: Initial Population Diversity and Genetic Algorithm Performance. In: Proc. of AIPR 2007. pp. 334–341 (2007)
8. Feldt, R., Poulding, S., Clark, D., Yoo, S.: Test set diameter: Quantifying the diversity of sets of test cases. In: Proc. of ICST (2016). pp. 223–233. IEEE (2016)

9. Fraser, G., Arcuri, A.: Whole test suite generation. IEEE Trans. Software Eng. 39(2), 276–291 (2013)
10. Jackson, D.: Promoting phenotypic diversity in genetic programming. In: Proc. of PPSN 2010. pp. 472–481. Springer (2010)
11. Maaranen, H., Miettinen, K., Penttinen, A.: On initial populations of a genetic algorithm for continuous optimization problems. J. Global Optim. 37(3), 405 (2007)
12. Mc Ginley, B., Maher, J., O'Riordan, C., Morgan, F.: Maintaining healthy population diversity using adaptive crossover, mutation, and selection. IEEE Trans. Evol. Comput. 15(5), 692–714 (2011)
13. McMinn, P.: Search-based software test data generation: a survey. Softw. Test. Verif. Reliab. 14(2), 105–156 (2004)
14. McPhee, N.F., Hopper, N.J.: AppGP: an alternative structural representation for GP. In: Proc. of CEC 1999. vol. 2, pp. 1377–1383. IEEE (1999)
15. Oliveto, P.S., Sudholt, D., Zarges, C.: On the benefits and risks of using fitness sharing for multimodal optimisation. Theor. Comput. Sci. 773, 53–70 (2019)
16. Palomba, F., Panichella, A., Zaidman, A., Oliveto, R., De Lucia, A.: Automatic test case generation: What if test code quality matters? In: Proc. of ISSTA (2016). pp. 130–141 (2016)
17. Panichella, A., Kifetew, F.M., Tonella, P.: Reformulating branch coverage as a many-objective optimization problem. In: Proc. of ICST 2015. pp. 1–10. IEEE (2015)
18. Panichella, A., Kifetew, F.M., Tonella, P.: Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. IEEE Trans. Software Eng. 44(2), 122–158 (2017)
19. Pétrowski, A.: A clearing procedure as a niching method for genetic algorithms. In: Proc. of ICEC 1996. pp. 798–803. IEEE (1996)
20. Robič, T., Filipič, B.: Differential evolution for multiobjective optimization. In: Proc. of EMO 2005. pp. 520–533. Springer (2005)
21. Rojas, J.M., Campos, J., Vivanti, M., Fraser, G., Arcuri, A.: Combining multiple coverage criteria in search-based unit test generation. In: Proc. of SSBSE 2015. pp. 93–108. Springer (2015)
22. Ronald, S.: Duplicate genotypes in a genetic algorithm. In: Proc. of CEC/WCCI 1998. pp. 793–798. IEEE (1998)
23. Rosca, J.P.: Entropy-driven adaptive representation. In: Proc. of Workshop on Genetic Programming: From Theory to Real-world Applications. vol. 9, pp. 23–32 (1995)
24. Sareni, B., Krahenbuhl, L.: Fitness sharing and niching methods revisited. IEEE Trans. Evol. Comput. 2(3), 97–106 (1998)
25. Shir, O.M.: Niching in evolutionary algorithms. In: Handbook of Natural Computing, pp. 1035–1070. Springer (2012)
26. Squillero, G., Tonda, A.: Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. Inf. Sci. 329, 782–799 (2016)
27. Toğan, V., Daloğlu, A.T.: An improved genetic algorithm with initial population strategy and self-adaptive member grouping. Comput. Struct. 86(11-12), 1204–1218 (2008)
28. Vogel, T., Tran, C., Grunske, L.: Does Diversity Improve the Test Suite Generation for Mobile Applications? In: Proc. of SSBSE 2019. pp. 58–74. Springer (2019)