Self-adaptation of Genetic Operators Through Genetic Programming Techniques

Andres Felipe Cruz-Salinas
Universidad Nacional de Colombia, Artificial Life Research Group (Alife)
afcruzs@unal.edu.co

Jonatan Gomez Perdomo
Universidad Nacional de Colombia, Artificial Life Research Group (Alife)
jgomezpe@unal.edu.co

ABSTRACT

Here we propose an evolutionary algorithm that self modifies its operators at the same time that candidate solutions are evolved. This tackles convergence and lack of diversity issues, leading to better solutions. Operators are represented as trees and are evolved using genetic programming (GP) techniques. The proposed approach is tested with real benchmark functions and an analysis of operator evolution is provided.

CCS CONCEPTS

•**Theory of computation** → **Bio-inspired optimization;** *Random search heuristics;* •**Mathematics of computing** → Probabilistic algorithms;

KEYWORDS

Evolutionary algorithms, real optimization, self-adaptation, genetic programming, self-adapted operators.

ACM Reference format:
Andres Felipe Cruz-Salinas and Jonatan Gomez Perdomo. 2017. Self-adaptation of Genetic Operators Through Genetic Programming Techniques. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071214

1 INTRODUCTION

Evolutionary algorithms (EAs) are metaheuristic optimization techniques inspired in biological evolution. A population of candidate solutions is maintained on each generation, and every candidate solution is encoded in an appropriate space in order to apply bio-inspired operators like selection, reproduction and mutation. A fitness function is defined in order to measure the quality of individuals. EAs present some issues that affect their performance: parameter tuning, premature convergence and lack of diversity.

Manual parameter tuning is the process of manually assigning parameter values to an EA. This process is, in general, tedious and time consuming. Parameter adaptation avoids the manual parameter tuning process and instead, values are modified by the algorithm according to certain rules that are predefined. For example, the one fifth rule controls the strength of the mutation according to its previous success[1]. In general, a static set of rules may work well in some kind of problems but are not general enough to work on other kind of problems.

Premature convergence is an issue that arises in population based strategies. Due to the pressure to obtain solutions that optimize a given problem, individuals converge quickly to local optima. An ad-hoc strategy is to increase the population size, but in a high dimensional problem it may cause a huge overhead. Another strategy is to force diversity in different ways, relax the pressure scheme or include new randomly-generated individuals. Those strategies compromise the population quality and may help to increase diversity, but the improvement on the best individual may not be significant. Finally, crowding and niching techniques make the population converge to different local optima at the same time, recombining individuals with similar mates in order to perform exploitation in different areas of the problem space.

Self-modifying operators attempt to tackle those issues (parameter selection, premature convergence and the lack of diversity) by changing the way individuals are generated according to the current population. Self-modification also provides individuals with higher quality due to a better exploration of the problem space.

Our proposal is an evolutionary algorithm where operators are defined as GP trees and are subject to evolution at the same time candidate solutions are evolved. This includes an additional source of diversity because the way the individuals is transformed changes along the algorithm execution. As a result of this diversity increase, the convergence of the algorithm is delayed and it leads to better results.

2 PREVIOUS WORK

In general, EAs use a fixed set of operators to be applied while evolving candidate solutions. These operators are inspired in biological evolution processes like reproduction of organisms, and have parameters that are usually tuned before running the algorithm[16].

There has been extensive work on self adapting the parameters at the same time the optimization process is carried on, especially in the continuous domain[13]. One of the most successful methods for continuous optimization is the CMA-ES [9] (Covariance Matrix Adaptation - Evolution Strategy), a widely applied strategy to solve real optimization problems which are non-linear and non-convex, especially when the objective function is ill-conditioned. There is also some research in self adapting parameters for combinatorial problems [19] [15]. Most of the work is focused on tuning numerical parameters of operators like mutation rates, and crossover points.

Recently, self adaptation has been done in specific types of problems, for example, a recent approach in [3] self-adapts the mutation operators guiding the search into the solution space using a self-adaptive reduced variable neighborhood search procedure in combinatorial problems. Another approach to solve multiobjective problems using self-adaptation is described in [8].

Finally, the approach described here is similar to ADF (automatically defined functions) proposed in [11] and [12]. As defined in [12]: An automatically defined function (ADF) is a function that is dynamically evolved during a run of genetic programming and that may be called by a calling program (or subprogram) that is concurrently being evolved. The main difference with the proposed approach, is that the evolved operators are meant to transform the individuals during the algorithm execution, whereas an ADF acts as reusable components that may be called by evolved programs in a genetic programming algorithm.

2.1 Parameter tuning

Manual parameter tuning is one of the most important aspects to consider in EAs. Due to the no free lunch theorem [18], there are no unique parameter values or even a unique algorithm that performs equally well for all optimization problems. An ad-hoc strategy is to perform the process manually, but it can be expensive and it is problem-dependent. This leads to automatic parameter tuning, where the researcher allows the algorithm to test multiple configurations of parameters and choose the one that works best. The F-Race and iterated F-Race methods [2] use statistical information for selecting the best configuration out of a set of candidate configurations under stochastic evaluations. In [14] there is a description of these methods as well as some improvements to the iterated racing method implemented in a software package called *irace*. Either manual or automatic, parameter tuning remains expensive, because the EA must be run in order to measure the effectiveness of a given configuration.

2.2 Parameter adaptation

Parameter adaptation (or control) is a strategy to find good parameter values without doing a manual search on every problem. The approach is to modify the parameter values at the same time the EA is searching for solutions. The ways to adapt the parameters are broad. There is an overview of techniques applied to numerical problems in [6]. A common strategy is related to the mutation rate, and another common approach described in [6], is to adapt through time the penalty of the fitness function, this is related to constrained optimization problems. Finally, [6] does a classification of the adaptation strategies in three categories: deterministic, adaptive and self-adaptive.

2.3 Adaptation through operator rates

There is a strategy that considers an EA in a higher level, it is to apply a single operator (from an operator set) on every generation depending on how good the operator is. This is achieved by associating an operator to a rate that measures its quality. In [7], there is a brief description of two rough categories of this approach: centralized and decentralized techniques.

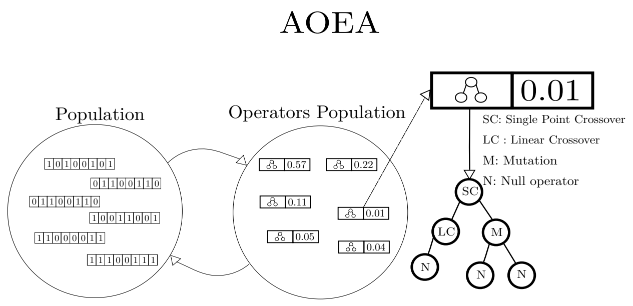


Figure 1: Graphical representation of the populations. In this case, the candidate solution encoding is binary. Each operator has two elements: A tree structure, and its probability to be chosen.

Finally, [7] proposes a hybrid approach by evolving the operator rates without using special metaoperators, the probability of choosing an operator at every iteration is either "punished" (decreased) or "rewarded" (increased), depending on the improvement of the individual when the operator is applied. As in decentralized techniques, the rates are normalized in order to sum one. The selection of the operator is a typical procedure (e.g., roulette or tournament) using the rates as the operator fitness.

3 PROPOSED APPROACH

The aim of this work is to go further from solely self-adapting the parameters in EAs towards self-modifying the structure of operators using GP techniques. This approach is a generalized version of HAEA [7]: the operators belong to an operators population and are exposed to evolution (like a coevolutionary technique). The strategy to select the operators is still the hybrid approach of [7], where a typical selection method (roulette or tournament) picks an operator proportionally to its probability to be chosen. From now on, AOEA (Adaptive Operators Evolutionary Algorithm) will refer to the proposed approach.

3.1 Operators as genetic programming trees

Here, the approach is to change the static structure of an operator and convert it into a genetic programming tree.

Atomic operator: An atomic operator is predefined by the user and is not exposed to evolution. It is one dimensional if defined as $o : D \longrightarrow D$ where D is the search space of the problem being solved. Similarly, an operator is said to be two dimensional if defined as $o : D^2 \longrightarrow D$. These operators capture the notion of mutation and crossover (respectively) in traditional genetic algorithms. A 1D operator takes a single individual as an argument and returns a modified version based on it (like classical mutation). Similarly, a 2D operator takes two individuals and produces a single individual, a "child" (this operation is not necessarily commutative). There is a special type of 2D atomic operator, called the *null* operator, which always return either the first or the second individual without modification.

Operator: An operator is defined as a binary tree, where each node contains either a 1D or a 2D atomic operator. The operator always receives two individuals as arguments, but it is up to the arity of the atomic operators the possible recombination of individuals. In order to compute the full transformation of an operator, a post-order traversal [4] is performed applying atomic operators transformations in a top-down fashion. A node performs an atomic operation with the arguments equal to the result of its children operators. This process continues recursively until a leaf is reached, in which case one of the two arguments is returned without modification (the *null* operator). The choice of which individual its returned is deterministic.

Formally, the operator O is defined as a triple:

$$O = [O_l, O_r, o]$$

Where O_l is the left child of O, O_r is the right child of O and o is an atomic operator. The result of an operator is computed as shown in equation 1

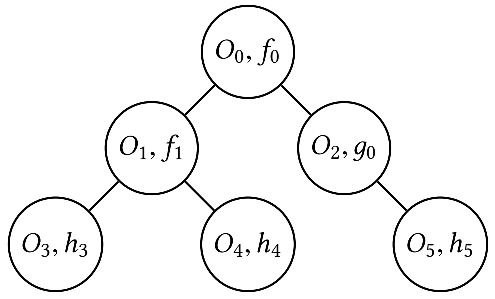
$$O(A,B) = \begin{cases} o(O_l(A,B), O_r(A,B)), & \text{if } \tau(O) = 2. \\ o(O_r(A,B)), & \text{if } \tau(O) = 1. \\ o(A,B), & \text{otherwise.} \end{cases} \quad (1)$$

$\tau(O)$ defines the number of children of a given node O. As in genetic programming, these trees are subject to mutation and recombination. Here, we use the following operators:

- **Mutation:** A mutation occurs on a random node of the tree, and the atomic operator of that node is randomly changed by an atomic operator of the same arity.
- **Recombination:** This meta operator takes two trees, selects a random node from each one, then swaps the subtrees from which the chosen nodes are roots. This method is described in [10].

Random node selection: In order to support the above operations a random node procedure must be defined. A reservoir sampling technique is applied in order to return a uniformly distributed node from the given tree.

The random operators population and the recombination/mutation procedures always guarantee that the number of children for a given node is consistent with the cardinality of its atomic operator. It is also guaranteed that the leaves always contain a *null* atomic operator. In the figure 2, there is a graphical example of an operator and how the atomic operators are composed in order to form the full operator.



$$O_0(A,B) = f_0(O_1(A,B), O_2(A,B))$$
$$O_1(A,B) = f_1(O_3(A,B), O_4(A,B))$$
$$O_3(A,B) = h_3(A,B) = A$$
$$O_4(A,B) = h_4(A,B) = B$$
$$O_2(A,B) = g_o(O_5(A,B))$$
$$O_5(A,B) = h_5(A,B) = B$$

Figure 2: Tree representation of an operator and its unfolded composition. The first element in each node represents its label, and the second element represents the atomic operator that belongs to that node. Below there is the operation performed on each node according to their children and their atomic operators. The atomic operators, labelled as f, g and h represent a two dimensional operator, a one dimensional operator, and the *null* operator, respectively.

3.2 Punish reward scheme

Based on HaEa[7], a punish reward scheme is defined in order to evolve individuals according to the operators quality. A operators population is defined with size κ. An operator's quality is represented by a number from the range [0, 1] (a rate), this measure is increased or decreased according to the performance of the operator on each generation. At the beginning of the algorithm, the quality measure is set to a random value. The operators population is evolved on each generation using the described recombination and mutation methods for genetic programming trees, and the selection is proportional to the quality measure previously described. The outline of the algorithm is presented in Algorithm 1.

On every iteration of the algorithm, an individual randomly selects one operator proportionally to the operators quality, and another individual is selected proportionally to its fitness value. These two individuals are recombined with the previously chosen operator, and the parent is replaced with its child if and only if the child fitness is equal or better. For each operator, there will be a vote system in order to measure the operator quality. If the fitness is better, there will be a positive vote, if it is worse, there will be a negative vote. This procedure is repeated for every individual in the population. Then, an operator will be rewarded if its vote count is positive, punished if it is negative. If it is zero, there will be no changes. After the rates are modified they are normalized.

Algorithm 1 AOEA outline. A Java implementation of this algorithm can be founded here https://github.com/afcruzs/AOEA.

```
1: function AOEA($\lambda$,$\kappa$)
2:     $t_0 = 0$
3:     $P_0 = initPopulation(\lambda)$
4:     $O_0 = initOperators(\kappa)$
5:     $R_0 = initRates(\kappa)$
6:     while not terminationCondition($P_t$) do
7:         crossoverPopulation($P_t$,$O_t$, $R_t$)
8:         crossoverOperators($P_t$,$O_t$,$\kappa$)
9:         mutationOperators($P_t$,$O_t$, $\kappa$)
10:        $t = t + 1$
11:    end while
12:    return best(P)
13: end function
```

The selection of individuals and operators is performed using a roulette selection algorithm. This recombination process is described in the algorithm 2.

Algorithm 2 Candidate solutions crossover

```
procedure crossoverPopulation($P_t$,$O_t$, $R_t$)
    $P_{t+1} = \{\}$
    $R_{t+1} = R_t$
    $V = initVotesToZero(\kappa)$
    for each ind $\in P_t$ do
        $operator = selectOperator(O_t, R_t)$
        $mate = selectIndividual(P_t)$
        $child1 = operator(ind, mate)$
        $child2 = operator(mate, ind)$
        $child = Best(child1, child2)$
        if fitness(child) > fitness(ind) then
            $V[operator] = V[operator] + 1$
        else
            $V[operator] = V[operator] - 1$
        end if
        if fitness(child) >= fitness(ind) then
            $P_{t+1} = P_{t+1} \cup \{child\}$
        else
            $P_{t+1} = P_{t+1} \cup \{ind\}$
        end if
    end for
    for each operator $\in O_t$ do
        $\delta = random(0, 1)$
        if V[operator] $> 0$ then
            $R_{t+1}[operator] = (1 + \delta) * R[operator]$
        else if V[operator] $< 0$ then
            $R_{t+1}[operator] = (1 - \delta) * R[operator]$
        end if
        normalizeRates(opRates)
    end for
end procedure
```

Then, the operators are recombined in order to evolve them proportionally to their quality using a roulette selection algorithm. Finally, a mutation on each tree is performed with probability equal to $1/\kappa$ as described above. These two processes are described in 3 and 4.

Algorithm 3 Operators crossover

```
procedure crossoverOperators($O_t$,$R_t$,$\kappa$)
    $O_{t+1} = \{\}$
    $shuffle(O_t)$
    for each $i$ where $i < \kappa$ and $i$ is even do
        $mate1 = O_{t,i}$
        $mate2 = O_{t,i+1}$
        $child1, child2 = recombine(mate1, mate2)$
        $O_{t+1} = O_{t+1} \cup \{child1, child2\}$
    end for
end procedure
```

Algorithm 4 Operators mutation

```
procedure mutationOperators($P_t$,$O_t$)
    $prob = 1.0/\kappa$
    for each operator $\in O_t$ do
        if random(0,1) <= prob then
            mutateOperator(operator)
        end if
    end for
end procedure
```

3.3 Operators initialization

The operators population is randomly generated before the execution of the algorithm, and in order to avoid too complex operators at the beginning every operator has a maximum depth of four. On each node, an atomic operator is chosen such that its arity is equal to the number of children. When the process reaches a leaf, a boolean flag is randomly generated to always return either the first or the second argument.

4 RESULTS

The proposed approach was tested with benchmark functions shown in table 1. These functions were selected because are standard on the real optimization literature, specially on evolutionary approaches. Each experiment is performed with 500 iterations, and with 50 and 100 individuals in the population. Additionally, the operators population κ is fixed to 16 on every experiment. The dimension for every experiment is set to 1000 unless the function is defined with a specific dimensionality. Every experiment is repeated 50 times. Finally, the initial population is randomly generated without violating the constraints of the function and it is the same for every experiment. The chosen coding method is a simple vector of real numbers.

The numerical results of the experiments are shown in tables 2 and 3. besides the proposed approach (AOEA), there is GA (genetic algorithm) and HAEA (Hybrid adaptive evolutionary algorithm from [7]). Those were implemented in order to have a baseline for comparison. In the GA the recombination method is linear crossover with random weights, the mutation operator is gaussian noise added to a random position of the chromosome. Finally, HAEA does not have parameters to tune besides the population size. For each iteration the best individual of each experiment is stored in order to visualize the convergence and compare the algorithms.

The atomic operators used in this experiment (for both HAEA and AOEA) are the following:

- swap two randomly chosen genes,
- add gaussian noise to a randomly chosen gene,
- single point crossover,
- uniform crossover,
- average crossover,
- linear crossover.

Figures 3, 4 and 5 contain some of the "hardest" functions of the tested dataset, the candidate solutions start far from the optimal value and the dimension is higher compared to the other functions. In general, the proposed strategy converges much later than the traditional GA and HAEA giving better results. The situation is

Table 1: Benchmark real functions. Every function has an optimal value of 0.0 except for H1 (is a maximizing function) which is 2. The functions are sorted in increasing "hardness". In general, the higher the dimension, the harder the function. On functions with the same dimension our measure of hardness is given by experimental results on how close are the results to the global optimum. The first column maps each function to an id to reference the functions on the results tables.

Id	Name	Function	Interval
1	Jong 1	$\sum_{i=1}^{N} x_i^2$	$x_i \in [-5.12, 5.12]$
2	Jong 2	$\sum_{i=1}^{N} (i+1)x_i^2$	$x_i \in [-5.12, 5.12]$
3	Jong 3	$\sum_{i=1}^{N} x_i^i$	$x_i \in [-1, 1]$
4	Himmelblau	$(x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$	$x_i \in [-6, 6]$
5	Two peak trap	$f(x) = \begin{cases} \frac{160}{15}(15-x) & \text{if } 10 \le x < 15 \\ \frac{200}{5}(x-15) & \text{otherwise} \end{cases}$	$x_i \in [-15, 15]$
6	Central two peak trap	$f(x) = \begin{cases} \frac{160}{15}x & \text{if } x < 10 \\ \frac{160}{15}(15-x) & \text{if } 10 \le x < 15 \\ \frac{200}{5}(x-15) & \text{otherwise} \end{cases}$	$x_i \in [-15, 15]$
7	H1	$\frac{sin(x_1 - \frac{x_2}{8})^2 + sin(x_2 + \frac{x_1}{8})^2}{\sqrt{(x_1-8.6998)^2+(x_2-6.7665)^2+1}}$	$x_i \in [-100, 100]$
8	Ackley	$20 - 20 * exp(-0.2 * \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}) + e - exp(\frac{1}{N}\sum_{i=1}^{N} cos(2\pi x_i))$	$x_i \in [-5, 5]$
9	Shubert 2D	$(\sum_{i=1}^{5} cos((i+1) * x_0 + i)) * (\sum_{i=1}^{5} cos((i+1) * x_1 + i))$	$x_i \in [-5.12, 5.12]$
10	Griewangk	$\frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} cos(\frac{x_i}{\sqrt{i}}) + 1$	$x_i \in [-600, 600]$
11	Rastrigin	$10 * N + \sum_{i=1}^{N} x_i^2 - 10 * cos(2\pi x_i)$	$x_i \in [-5.12, 5.12]$
12	Schaffer	$\sum_{i=1}^{N-1}(x_i^2 + x_{i+1}^2)^{0.25} * [sin(50 * (x_i^2 + x_{i+1}^2)^{0.1}) + 1]$	$x_i \in [-100, 100]$
13	Rosenbrock	$\sum_{i=1}^{N-1} 100 * (x_{i+1} - xi^2) + (1 - x_i)^2$	$x_i \in [-2.048, 2.048]$
14	Bohachevsky	$\sum_{i=1}^{N-1}(x_i^2 + 2x_{i+1}^2 - 0.3cos(3\pi x_i) - 0.4cos(4\pi x_{i+1}) + 0.7)$	$x_i \in [-100, 100]$
15	Schwefel	$418.9829 * n \sum_{i=1}^{n} x_i * sin(\sqrt{\lvert x_i \rvert})$	$x_i \in [-500, 500]$

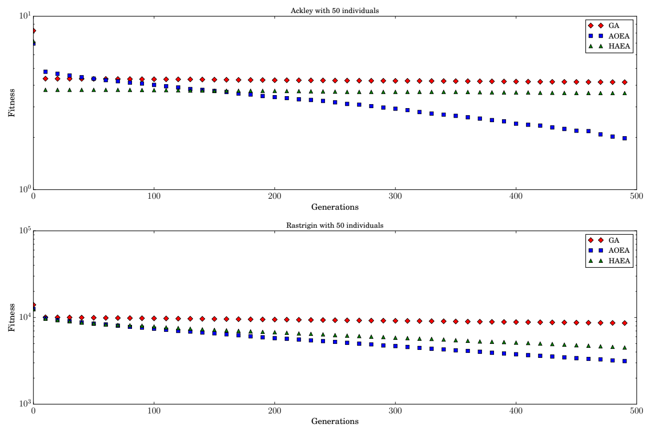


Figure 3: Median of Ackley and Rastrigin functions with 50 individuals in the population. The fitness is on logarithmic scale.

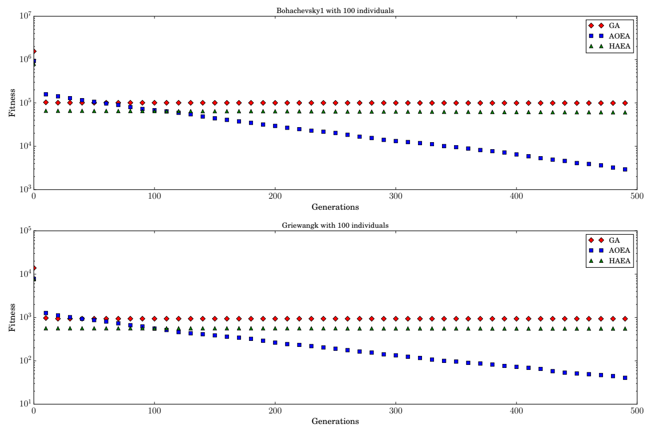


Figure 4: Median of Bohachevsky and Griewangk functions with 50 individuals in the population. The fitness is on logarithmic scale.

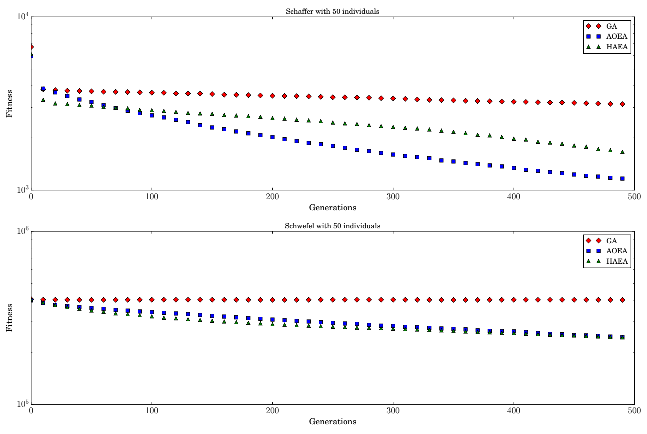


Figure 5: Median of Schaffer and Schwefel functions with 50 individuals in the population. The fitness is on logarithmic scale.

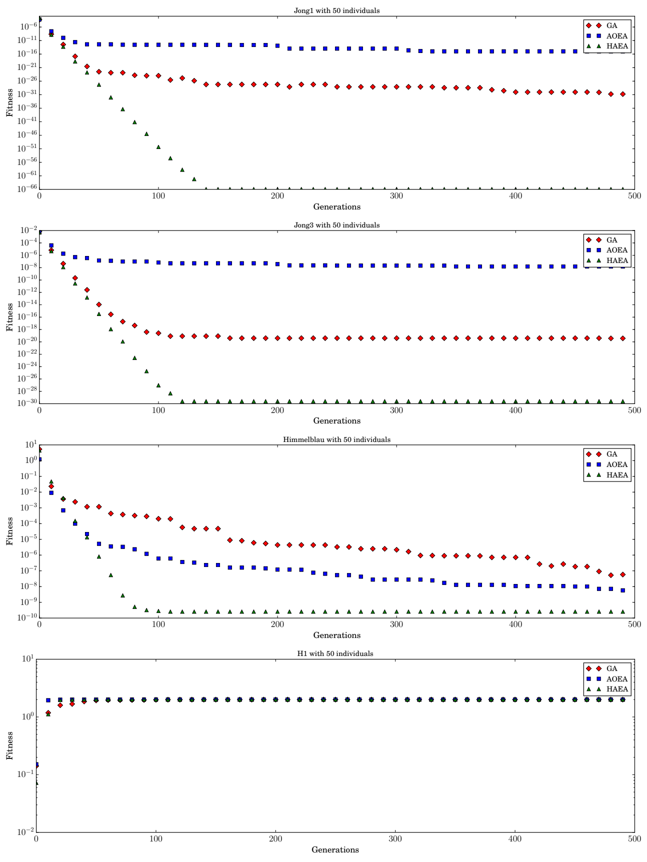


Figure 6: Functions with low dimensionality: Median of Jong1, Jong3, Himmelbau and H1. With 50 individuals in the population. The fitness is on logarithmic scale.

Table 2: Results of the last generation with 50 individuals in the population. Best results are in bold.

Id	GA median	HAEA median	AOEA median
1	1.6E-31 ± 0.0	1.4E-66 ± 0.0	**5.4E-70 ± 0.0**
2	1.9E-35 ± 0.0	5.7E-78 ± 0.0	**2.3E-81 ± 0.0**
3	3.6E-20 ± 0.0	2.4E-30 ± 0.0	**9.0E-45 ± 0.0**
4	5.8E-08 ± 0.0	2.5E-10 ± 0.0	**1.1E-11 ± 0.0**
5	**0.0E+00 ± 0.0**	5.3E-03 ± 0.010	1.7E-04 ± 0.0
6	**0.0E+00 ± 0.0**	4.1E-03 ± 0.010	9.6E-05 ± 0.0
7	**2.0E+00 ± 0.3**	**2.0E+00 ± 0.001**	**2.0E+00 ± 0.0**
8	4.1 ± 0.1	3.5 ± 0.3	**7.8E-01 ± 0.185**
9	**1.8E+02 ± 21.6**	**1.8E+02 ± 0.0**	**1.8E+02 ± 0.0**
10	1.8E+03 ± 279.5	1.0E+03 ± 1.3E+03	**2.2E+01 ± 8.2**
11	8.6E+03 ± 152.4	4.4E+03 ± 6E+02	**1.4E+03 ± 167.6**
12	3.1E+03 ± 166.2	1.6E+03 ± 283.3	**1.1E+03 ± 57.1**
13	1.0E+04 ± 1.2E+03	6.2E+03 ± 7.1E+03	**1.2E+03 ± 78.923**
14	1.8E+05 ± 2.5E+04	1.0E+05 ± 1.6E+05	**7.5E+02 ± 1282.9**
15	4.0E+05 ± 3220.7	2.4E+05 ± 7648.8	**8.5E+04 ± 1.7E+04**

Table 3: Results of the last generation with 100 individuals in the population. Best results are in bold.

Id	GA median	HAEA median	AOEA median
1	1.288E-57 ± 0.0	**8.349E-266 ± 0.0**	1.762E-173 ± 0.0
2	2.0E-57 ± 0.0	**1.5E-265 ± 0.0**	1.5E-174 ± 0.0
3	4.0E-46 ± 0.0	**1.9E-138 ± 0.0**	4.0E-92 ± 0.0
4	2.4E-10 ± 0.0	6.2E-14 ± 0.0	**6.0E-24 ± 0.0**
5	**0.0E+00 ± 0.0**	1.9E-03 ± 0.005	5.8E-05 ± 0.0
6	**0.0E+00 ± 0.0**	3.0E-03 ± 0.006	5.4E-05 ± 0.0
7	**2.0E+00 ± 0.001**	**2.0E+00 ± 0.0**	**2.0E+00 ± 0.0**
8	3.4E+00 ± 0.1	3.0E+00 ± 0.4	**4.0E-01 ± 0.0**
9	**1.8E+02 ± 0.005**	**1.8E+02 ± 0.0**	**1.8E+02 ± 0.0**
10	9.3E+02 ± 93.2	5.5E+02 ± 519.15	**8.2E+00 ± 2.2**
11	8.2E+03 ± 146.6	4.3E+03 ± 956.0	**7.9E+02 ± 86.6**
12	2.6E+03 ± 75.5	1.5E+03 ± 292.1	**8.7E+02 ± 48.9**
13	5.5E+03 ± 532.9	3.5E+03 ± 1420.4	**1.058E+03 ± 17.2**
14	9.9E+04 ± 7621.1	6.0E+04 ± 62534.6	**1.6E+02 ± 530.4**
15	4.0E+05 ± 1635.1	1.817E+05 ± 7809.3	**3.6E+04 ± 11764.7**

a bit different in the set of functions where the dimensionality is fixed to two (figure 6): in those cases solutions are very close to an optimal value of 0.0 and those cases HAEA generally performs better than the proposed approach. Nevertheless, the solutions of almost every algorithm are very good because the analytic form of the functions is simple and the dimensionality is two compared to 1000 on the "harder" functions.

4.1 Statistical tests

The proposed approach is compared to the baseline algorithms (classic GA and HAEA [7]) by applying a statistical test over the best individual (on each experiment) in the last generation on every objective function and population size. Initially, the measurements were tested using a D'Agostino's K-squared normality test. Only about 45% of all the experiments passed the test, so a Wilcoxon signed-rank test was used with the null hypothesis that the paired

Table 4: Experiments on which the null hypothesis was not rejected. AOEA vs GA.

Function	Population	Positive sum	Negative sum	W
Himmelblau	100	794.0	481.0	481.0
Shubert2D	100	811.0	464.0	464.0
Shubert2D	50	775.0	500.0	500.0

Table 5: Experiments on which the null hypothesis was not rejected. AOEA vs HAEA.

Function	Population	Positive sum	Negative sum	W
H1	100	587.0	688.0	587.0
Himmelblau	100	444.0	831.0	444.0
Himmelblau	50	654.0	621.0	621.0
Schwefel	50	639.0	636.0	636.0

samples come from the same distribution with a 95% confidence interval.

The results shown in 4 and in 5 confirm the previous intuition. The proposed approach is not statistically different *only* in some functions that have lower dimensionality and simpler analytic form (with the exception of the Schewfel function). In the other functions, the proposed approach gives a better performance and in some functions the algorithm did not converge in the 499th iteration, with more fitness evaluations is expected to obtain even better results.

4.2 Analysis of operators behaviour

Trees similarity: The trees are stored per generation and are compared pairwise using the tree edit distance proposed by Zhang and Shasha [20], this distance is the minimum number of operations to transform a tree into another tree. On each generation, the trees are transformed into a two dimensional space for visualization using multidimensional scaling with a the matrix of (normalized) pairwise distances. The graphical results are presented in figure 7. The trees tend to converge to a single cluster, but they never group into a single one. The behaviour is to converge until certain point then start to separate and then group again and so on. In the last plot of figure 7 there is a snapshot of the 499th generation where the operators have not converged yet.

The trees never converge into a very similar tree, which is good because diversity is maintained, and it implies that different changes are applied to the candidate solutions, giving them the chance to delay the convergence but still produce good results. Another interesting result is the behaviour of the operator rates, where, once again, the dynamics change according to the "hardness" of the tested function. Figure 8 shows the maximum rate (probability) of the operators population per generation.

From figure 8 it can be seen, that the best rates per generation do not converge right away but "oscillate" through the iterations of the algorithm. Whenever a set of rates is very high, the complement is going to have a very low probability to be selected because the selection method is proportional to the rates. The fact that the rates

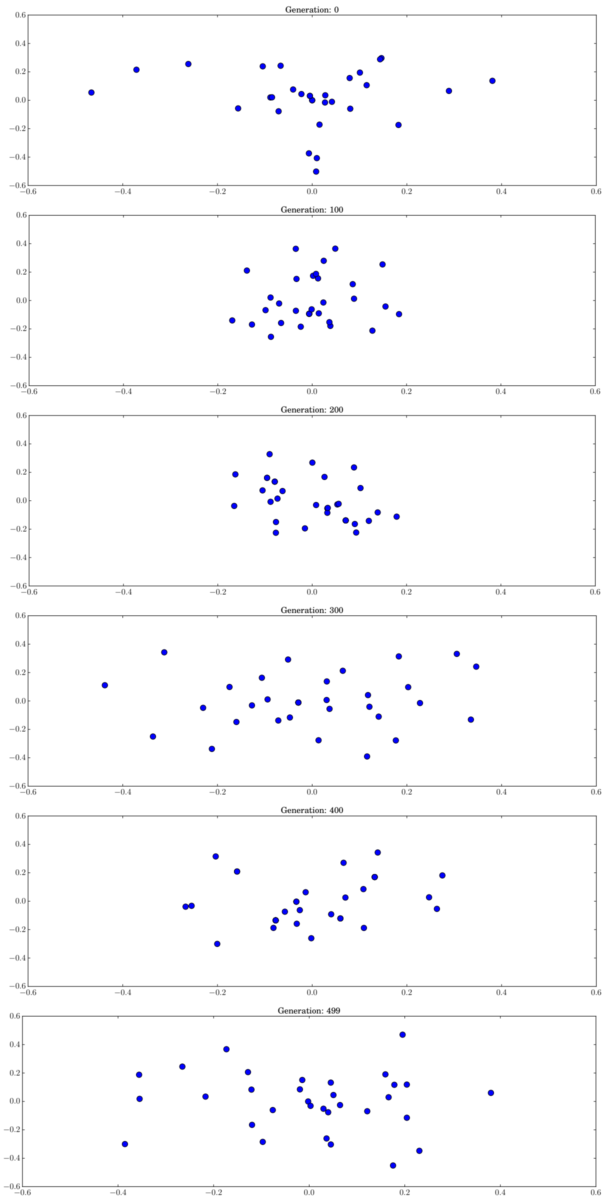
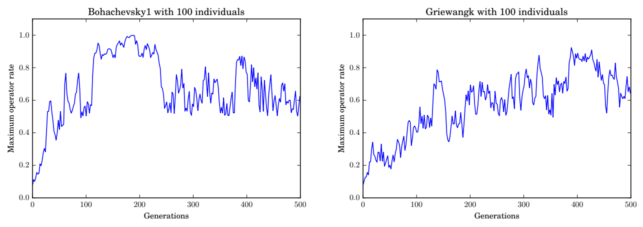
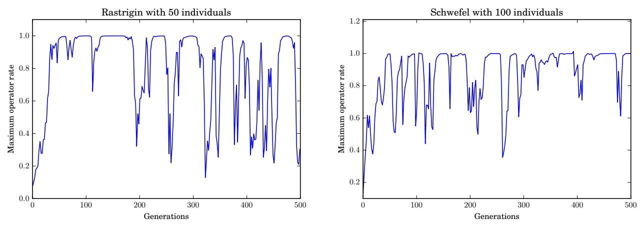


Figure 7: Trees embedded in a 2D space in generation from 0 to 499. The operators are evolved to minimize the Ackley function. The magnitude of the coordinates is a result of the Multidimensional Scaling process which enforces the points to be close if their tree distance is small. The 2D embedding was computed using the sklearn [17] implementation of the SMACOF (Scaling by Majorizing a Complicated Function) algorithm [5].

do not converge to a rate close to 1.0 means that operators that are applied to individuals are not always the same. Furthermore, the rates are also used to evolve operators, which contributes to maintain the overall diversity as was shown in the 2D embedded visualizations. These are empirical conclusions given the patterns



(a) Best rates of Bohachevsky, and Griewangk function with 100 individuals in the population.



(b) Best rates of Rastrigin, Schwefel, function with 50 and 100 individuals in the population.

Figure 8: Maximum rates over operators population

revealed by the data, more rigorous analysis using statistical tools is out of the scope of this work.

5 CONCLUSIONS AND FUTURE WORK

The proposed algorithm shows a fast convergence in most of the functions tested, and does not fall into premature convergence due to the generated diversity by the operators scheme and the punish/reward update, which puts enough pressure to achieve desirable results. Moreover, this scheme is easily applied to other kinds of problems without having to specify complex operators to generate new solutions, but only defining small atomic operators (which incorporate knowledge of the problem domain) and let the evolutionary process combine them. Finally, in numeric optimization problems there is no previous knowledge required about the function like gradients or the specific function, but in order to obtain better results and fast convergence it is useful to know the constraints for each dimension in order to maintain feasible solutions along the algorithm. It should be noted that there is an runtime overhead on the operators evaluation, as well on its selection according to its quality measure. However, this overhead remains constant with respect to the number of fitness evaluations, which is usually the bottleneck on real-world applications. Further more, in our experiments GA and HAEA converged quickly to bad local minima. Due to the selection pressure it can be hypothesized that with more computational resources they will not evolve better solutions because the diversity is greatly reduced.

Future work includes applying this approach in other problems outside of the numerical optimization domain and possibly in other contexts like open ended evolution, non stationary functions, and multi-objective optimization, where self-adaptation in the breeding operators is needed in order to maintain genetic diversity. As usual with evolutionary strategies, the population size has a crucial role on maintaining diversity. Future work, also includes finding a way to self-adapt the population size with techniques related to the proposed approach, as well as applying this approach with separately evolved, small operator populations for every candidate solution that exchange information between each other.

Finally, we hope to do a more rigorous analysis of the operators convergence using appropriate statistical tests and more refined techniques to compare the trees structure and its relation with the problem being optimized.

REFERENCES

[1] Beyer, H.G., Schwefel, H.P.: Evolution strategies–a comprehensive introduction. Natural computing 1(1), 3–52 (2002)
[2] Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated f-race: An overview. In: Experimental methods for the analysis of optimization algorithms, pp. 311–336. Springer (2010)
[3] Coelho, V., Coelho, I., Souza, M., Oliveira, T., Cota, L., Haddad, M., Mladenovic, N., Silva, R., Guimarães, F.: Hybrid self-adaptive evolution strategies guided by neighborhood structures for combinatorial optimization problems. Evolutionary computation 24(4), 637–666 (2016)
[4] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms second edition (2001)
[5] De Leeuw, J., Mair, P.: Multidimensional scaling using majorization: Smacof in r. Department of Statistics, UCLA (2011)
[6] Eiben, Á.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on evolutionary computation 3(2), 124–141 (1999)
[7] Gomez, J.: Self adaptation of operator rates in evolutionary algorithms. In: Genetic and Evolutionary Computation Conference. pp. 1162–1173. Springer (2004)
[8] Hadka, D., Reed, P.: Borg: An auto-adaptive many-objective evolutionary computing framework. Evolutionary computation 21(2), 231–259 (2013)
[9] Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on. pp. 312–317. IEEE (1996)
[10] Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection, vol. 1. MIT press (1992)
[11] Koza, J.R.: Genetic programming ii: Automatic discovery of reusable subprograms. Cambridge, MA, USA (1994)
[12] Koza, J.R., Andre, D., Bennett III, F.H., Keane, M.A.: Use of automatically defined functions and architecture-altering operations in automated circuit synthesis with genetic programming. In: Proceedings of the 1st annual conference on genetic programming. pp. 132–140. MIT Press (1996)
[13] Kramer, O.: Evolutionary self-adaptation: a survey of operators and strategy parameters. Evolutionary Intelligence 3(2), 51–65 (2010)
[14] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58 (2016)
[15] Maruo, M.H., Lopes, H.S., Delgado, M.R.: Self-adapting evolutionary parameters: encoding aspects for combinatorial optimization problems. In: European Conference on Evolutionary Computation in Combinatorial Optimization. pp. 154–165. Springer (2005)
[16] Mitchell, M.: An introduction to genetic algorithms. MIT press (1998)
[17] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011)
[18] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE transactions on evolutionary computation 1(1), 67–82 (1997)
[19] Younes, A., Basir, O., Calamai, P., Areibi, S.: Adapting genetic algorithms for combinatorial optimization problems in dynamic environments. INTECH Open Access Publisher (2008)
[20] Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM journal on computing 18(6), 1245–1262 (1989)