Improving Rule Based and Equivalent Decision Simplifications for Bloat Control in Genetic Programming Using a Dynamic Operator

Gustavo F.V. de Oliveira and Marcus H.S. Mendes

ICET, Universidade Federal de Viçosa, Florestal, Brazil
{gustavo.viegas,marcus.mendes}@ufv.br

Abstract. Bloat is a common issue regarding Genetic Programming (GP), specially noted in Symbolic Regression (SR) problems. Due to this, GP tends to generate a huge amount of ineffective code that could be avoided or removed. Code editing is one of many approaches to avoid bloat. The objective in this strategy is to mutate or remove subtrees which do not contribute to the final solution. Two known methods of redundant code removal, the Rule Based Simplification (RBS) and Equivalent Decision Simplification (EDS) are extended in a new operator presented in this paper, called Dynamic Operator with RBS and EDS (DORE). This operator gives the algebraic simplification table used by RBS the potential to learn from reductions performed by EDS. An initial benchmark highlighted how the RBS table can grow as much as 86% with DORE, and reducing the time spent on simplification by 16.83%. Experiments with the other three SR problems were performed showing a considerable improvement on fitness of the generated programs, besides a slight reduction in the population of the average tree size.

Keywords: Genetic Programming · Bloat Control · Code Editing.

1 Introduction

Symbolic Regression (SR) is one of the main applications and motivators to Genetic Programming (GP), a method for automatically generate computer programs from a high level definition of a problem [15]. Since early 90ś, many data-driven problems are modeled as SR problems [2], where no previous knowledge or pre-processing input is required. GP is well suited for resolving such problems since any algebraic function set can be effectively represented as trees and implemented as computer programs for the problem domain [9].

Bloat - the uncontrolled and excessive growth of individuals without a proportional gain of fitness - is a well-known issue and a field of study in GP. It is specially noted in SR. The large amount of inefficient code causes excessive consumption of computational resources, as well as many other practical issues [16], hiding the problems real complexity and domain.

There are many approaches for avoiding uncontrolled code growth in tree-based GP [5], some of which are presented as follows. The most simple and

popular of them is implementing a Depth Limit [9], although is not a truly effective approach, as it can induce growth in some scenarios. Parsimony Pressure [9, 13] is another popular technique which adds a penalty term in fitness function to punish large trees. Pseudo-hill Climbing [6] attempts to guarantee the fitness to improve in population rejecting individuals least fit than its parents until one is finally accepted. Code Editing approach mutates or removes redundant sub-trees in individuals.

Regarding the code editing approach, Koza [9] proposed a simple method to simplify (to convert a tree into a smaller, equivalent, tree) using grammar rewrite rules, which further inspired the Rule Based Simplification (RBS) method [7, 20, 11]. RBS was extended with Equivalent Decision Simplification (EDS) [11], which recursively compares all sub-trees in an individual for equivalency with a small set of terminals.

This paper proposes the Dynamic Operator with RBS and EDS (DORE), which improves a code editing bloat control algorithm using both RBS and EDS to maximize its reduction potential without greater punishments in execution time. The main feature of DORE is to dynamically learn redundant expressions to be applied with RBS from EDS outputs. It also optimizes the access of RBS rules using the *any* keyword in a hash-table implementation, as well as introduces a warm-up stage to grow RBS rules before the evolutionary process begins.

The article is organized as follows: Section 2 provides a quick overview about code editing operators; In Sect. 3 an improved strategy using the previous operators is proposed; Sect. 4 shows how RBS and EDS were implemented and the technical resources utilized in benchmarks; Sect. 5 shows a performance comparison between the simplification operators, beside the empirical results and discussion over three SR benchmark problems; and finally in Sect. 6, the conclusion is presented and the possibilities for further research.

2 Background

Redundancy is one of the key contributors to inefficient code growth. EDS [11] was introduced to extend RBS as they complement each other. RBS removes redundant subtrees by replacing a tree for a smaller equivalent one by applying arithmetic rules such as $X/1 \rightarrow X$ and $0*X \rightarrow 0$. These rules must be known and provided before the evolutionary process starts, thus each rule must be explicitly specified. Another limitation by RBS is that it's rules must be specified exactly like it would appear in an individual. For example, the rule $X*0 \rightarrow 0$ would not be sufficient to simplify a tree $0*X$ unless the rule $0*X \rightarrow 0$ is specified.

EDS extends RBS in a manner that it simplifies trees without previous knowledge of algebraic rules. It can also remove redundancies that are only true in the training domain. The simplification by EDS is made by evaluating each subtree in an individual and comparing these subtrees with a set of small trees or terminals which is usually the result of simplifications, such as X, 1 and 0. In a SR problem, EDS is evaluated as follows [11]:

1. Determine a suitable set of simple trees S_{simple}.

2. Check all subtrees in the target for equivalence to a tree in S_{simple}.
3. If some subtree is equivalent to a tree in S_{simple}, and larger than it, replace that subtree with the simple tree.
4. Repeat this procedure recursively until it fails.

Finding a suitable set of trees S_{simple} for a problem is not an easy task. Usually the set of terminals is a natural fit for S_{simple}, but if it is already known that some subtrees must appear in the final output, such as in trigonometrical problems, they can be inserted in this set. However, the main issue of using EDS as a single operator of bloat control is the computational performance. With problems hard enough and individuals big enough, reducing a single generation can take as long as multiple generational evaluations. This scenario is not uncommon in simple SR problems. If an evaluated individual with EDS contains more subtrees than the population size, the evaluation function will be called upon as much as in the generational evaluation.

2.1 Simplification with RBS and EDS

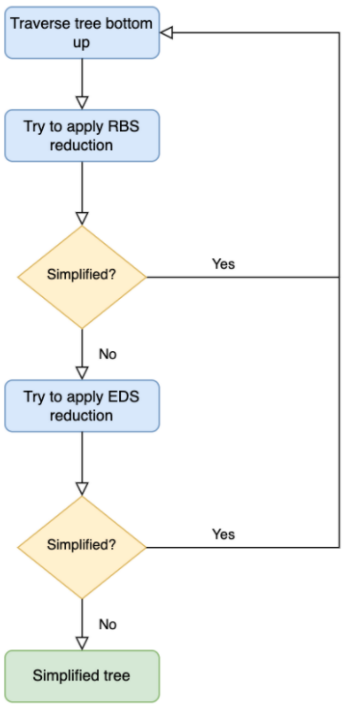

Fig. 1. Flow of simplification using RBS and EDS

The flow of simplification using both RBS and EDS [11] is shown in Figure 1. It can be expressed as follows:

1. Let the genotype tree of individual i be t_i.
2. Apply RBS recursively to all nodes of t_i, until there is no node to which RBS can be applied, obtaining t'_i.
3. Apply EDS to all nodes of t'_i. If any node is translated, let the translated tree be t_i and go to (2). If there is no node to which EDS can be applied, finish and let t'_i be the final result.

Applying RBS before applying EDS is a good idea since it prevents EDS to be used for simplifying already known rules. Also, RBS execution time is lower than EDS, which will be explored further in this paper.

3 Proposed Improvements

The main idea of the improvements to the simplification with RBS and EDS flow is to increase RBS rules table R dynamically as soon as new rules, or more efficient ones, are discovered. This allows RBS to execute independently with no large impacts in execution time. A more robust RBS operator also helps in simplification itself, since any simplification made by EDS triggers new RBS calls, as well as new EDS calls. If simplifications made by EDS occurred in the first step of the simplification flow, then it would eliminate the need to reevaluate subtrees that would only be reduced by EDS.

The first improvement proposed is to allow the simplification rules table R, used by RBS, to have rules inserted in execution time. Each time a subtree is simplified by RBS using the keyword $\_ANY\_$, which denotes any subtree, a new rule with the original subtree as input is inserted in R. Afterwards, new calls to RBS with the same subtree as input have $O(1)$ access guaranteed in a hash-table implementation.

Analogously, each simplification made with EDS creates a new rule in R table with the learned simplification. Figure 2 shows how the previous flow is extended with an additional step with for learning new RBS rules. This way, there is no need to apply simplification with EDS to the same subtree in subsequent individuals. Algorithm 1 shows an example algorithm to simplify a generation using this improved simplification flow with RBS and EBS in a SR problem.

Another strategy adopted to grow even more the R table was the insertion of a warm-up step before the evolutionary process begins. This warm-up step consists of generating random trees in the training domain of a problem, with random sizes, and then simplifying them with the improved flow. Each successful reduction creates a new rule in the table, so the GP starts with a larger R table.

4 Methodology

All algorithms were implemented in Python 3.8.3 with the package DEAP [4] (Distributed Evolutionary Algorithm in Python) in version 1.3.1. Parallelism was

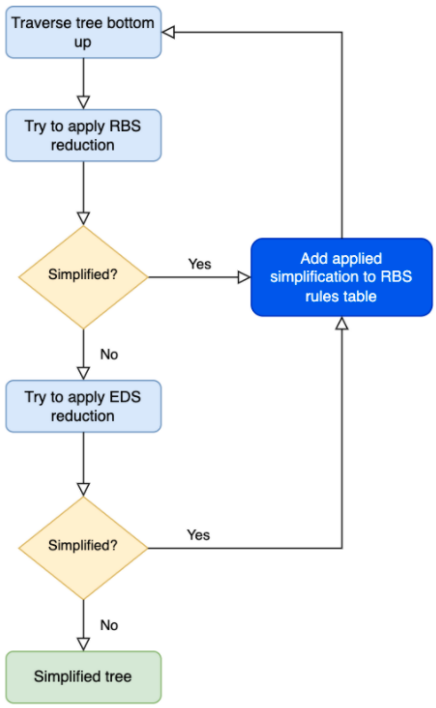

Fig. 2. Flow of simplification using RBS and EDS with the proposed improvements

used to evaluate fitness functions using the package Ray [12] in version 0.8.6. The benchmarks were conducted in a 2018 MacBook Pro with 2.2 GHz 6-Core Intel Core i7 and 16 GB 2400 MHz DDR4 memory.

4.1 Rule Based Simplification

The RBS simplification rules table R was implemented as a hash table. The keyword $\_ANY\_$ denotes any subtree and allows rules such $\_ANY\_ * 0 \rightarrow 0$ to be defined. In the original paper [11] this keyword was referenced as A and no further details were provided on how it was implemented. In this paper, every subtree evaluated by RBS consults the hash table directly and only if no rule with the subtree is described the $\_ANY\_$ rules are consulted. The access to the table R has complexity $O(1)$, except on the scenario where the $\_ANY\_$ keyword is included in the rule body, which makes the complexity $O(N)$ where N is the amount of rules in the table.

Reducing the amount of rules with $\_ANY\_$ is a slight improvement to this implementation as it maximizes the rate of $O(1)$ accesses. Another way to attain this goal is to define as many redundant rules as possible in R, for example prioritizing the definition of rules $1 * 0 \rightarrow 0$ and $X * 0 \rightarrow 0$ instead of $\_ANY\_ *$

Algorithm 1: Example of a generation simplification using improved RBS and EDS flow

```
Data: P: generation population; R: RBS rules table
population_simplified ← {}
foreach individual I in P do
    foreach subtree I_subtree in I do
        subtree_simpl, simplification ← call RBS with I_subtree
        if simplification exists and not in R then add simplification to R ;
        subtree_simpl, simplification ← call EDS with subtree_simpl
        if simplification exists then
            if simplification not in R then add simplification to R ;
            go back to the beginning of current loop
        end
        I_subtree ← subtree_simpl
    end
    add I to population_simplified
end
return population_simplified
```

$0 \rightarrow 0$. This task can be hard when there is no previous knowledge of such simplifications in the domain of the problem.

4.2 Equivalent Decision Simplification

The subtrees set S_{simple} for comparison for equivalence in EDS was implemented as a simple list. Two subtrees is considered equivalent if all fitness values elapsed from this subtree in the domain has relative error $\epsilon < 0.005$. This threshold was chosen after some tests and set to all benchmark problems in both compared operators in this paper.

5 Experimental Results

This section is organized as follows: Section 5.1 shows a preliminary benchmark, validating the performance improvement on each simplification method; It is presented in Sect. 5.2 the results for three artificial SR problems using the proposed operator, comparing it with the original operator and baseline GP.

5.1 Simplification Methods Performance

A simple benchmark - using $cos(2\pi x)$ as objective function - was ran 50 times to compare the execution time of the five simplification strategies.

Experimental Setting The experiment was the simplification of many independent individuals in each reduction strategy: traditional versions of RBS and EDS, RBS and EDS with dynamic rules allowed and RBS after the warm-up step. More details of each strategy were discussed in Section 3. The fitness function was defined as the RMSE (Root Mean Squared Error), of 20 uniform points in the interval $[-\pi, \pi]$. The function set used was $\{+, -, *, \div\}$, where $\div$ is the protected division satisfying $X/0 \to 1$. The terminal set is $\{X, 0, 1, \pi\}$. For each execution, 200 trees are generated and then cloned in each step to avoid any bias or propagation of simplifications. The reductions applied in one step are not carried on to the next. The procedure, shown in Figure 3, is described as:

1. Reduction with RBS is applied to the 200 cloned individuals.
2. Reduction with EDS is applied to the 200 cloned individuals.
3. With the dynamic insertions in the rules table R allowed, reduction with RBS is applied to the 200 cloned individuals.
4. With the dynamic insertions in the rules table R allowed, reduction with EDS is applied to the 200 cloned individuals.
5. With the dynamic insertions in the rules table R allowed, a warm-up step is done, and reduction with EDS is applied to the 200 cloned individuals.

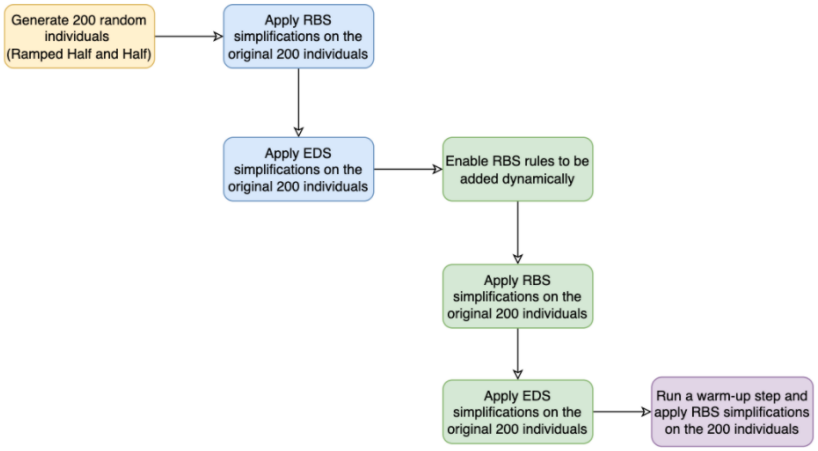

Fig. 3. Flow for the simplification methods performance comparison benchmark

The 200 individuals are generated in each of 50 executions with the method **Ramped Half and Half** and with depth limited to $[5, 10]$. The warm-up step applied RBS and EDS sequentially to 200 trees generated with **Grow** method and with depth limited to $[1, 5]$. The initial RBS rules table R is defined in Table 1.

$$\_ANY\_ + 0 \to \_ANY\_,\ \_ANY\_ + 0 \to \_ANY\_,\ \_ANY\_ * 1 \to \_ANY\_,$$
$$1 * \_ANY\_ \to \_ANY\_,\ \_ANY\_ * 0 \to 0,\ 0 * \_ANY\_ \to 0,$$
$$\_ANY\_ - \_ANY\_ \to 0,\ 1 - 1 \to 0,\ 0 - 0 \to 0,\ 0 + 0 \to 0,$$
$$\_ANY\_ - 0 \to \_ANY\_,\ 1 \div 0 \to 1,\ 0 \div 1 \to 0,\ 0 \div 0 \to 1,\ 1 \div 1 \to 1,$$
$$\_ANY\_ \div 0 \to 1,\ 0 \div \_ANY\_ \to 0,\ \_ANY\_ \div \_ANY\_ \to 1,$$
$$\_ANY\_ \div 1 \to \_ANY\_$$

Table 1. Initial simplification rules table used by RBS

Results and Discussion Table 2 shows the arithmetic mean of the 50 executions in each strategy. Dynamic RBS is 4.84% faster than the default RBS. The RBS after the warm-up is shown to be 16.83% faster than the traditional RBS and provided a 11.52% improvement to dynamic RBS before the warm-up.

Simplification Strategy	RBS	EDS	Dynamic RBS	Dynamic EDS	Dynamic RBS after warm-up
Avg. Execution Time (s)	5.49	11.48	5.23	9.61	4.69

Table 2. Execution time comparison of the simplification operators

This benchmark indicates that not only a bigger R table can improve the simplifications performance but also how much slower EDS is compared to RBS as well. With no improvements, EDS is 2.09 times slower than RBS. Dynamic EDS is 1.84 times slower than dynamic RBS before warm-up. Dynamic RBS after the warm-up can be twice as fast as the dynamic EDS. The warm-up increased the rules of table R, in average, from 19 entries to 135, to an around 86% improvement.

5.2 Symbolic Regression Problems

To validate the proposed improvements, 3 simple artificial SR problems were tested. These benchmarks were extracted in [5], selected from [8, 10, 17–19]. Since this paper presents a preliminary analysis of the operator, the benchmark problems are rather simple and harder problems will be explored in future works. The problems are presented in Table 3.

Experimental Setting For each problem, 50 independent executions were performed, in 3 different approaches: **Base GP**: with no code editing bloat control operator, **Baseline Operator**: with the original simplification flow of RBS and EDS as in [11], and **DORE**: using the proposed dynamic operator.

The function set to all benchmarks is set to $\mathcal{F} = \{+, -, *, \div\}$ where $\div$ is the protected division satisfying $X/0 \to 1$. The terminal set to benchmarks 1 and

Benchmark	Objective Function	Function Formula	Domain - Training
1	$f_1(x_1, x_2, x_3, x_4, x_5)$	$\frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$	50 random points. $x_i \in [0, 6]$
2	$f_2(x_1, x_2)$	$\frac{(x_1-3)^4+(x_2-3)^3+(x2-3)}{(x_2-2)^4+10}$	50 random points. $x_i \in [0, 6]$
3	$f_3(x)$	$0.3\ x\ sin(2\pi x)$	40 random points. $x \in [-2, 2]$

Table 3. Artificial symbolic regression problems

2 is $\tau = \{0, 1\}$ and the decision variables. For benchmark 3, the terminal set is $\tau = \{0, 1, \pi\}$ and the decision variable. To all benchmarks, the fitness function is the RMSE of the training domain points.

Besides the traditional GP parameters, two new parameters were introduced to fit the operator's context in more computational resource expansive problems. The first one is the percentage of population ρ which passes through the full simplification flow. The algorithm is developed in such way that the best $\rho\%$ individuals of the population are chosen each generation. The second additional parameter is the remaining best γ percentage of the population that was not previously chosen and passes through the reduction with RBS only. For example, if the population size is 100, and the values $\rho = 40\%$ and $\gamma = 20\%$, the 40 most fit individuals would pass through the complete reduction flow and the remaining best 20 trees would be reduced with RBS only.

These additional parameters are needed due the high computational cost of the reductions that would make unfeasible the operation of all individuals without a great impact in execution time. A large γ value does not have a high impact on the execution time, since RBS reduction is simpler. Nonetheless, the ρ value greatly impacts the total execution time. For the benchmarks these parameters were chosen empirically, testing values big enough that would not make the total GP execution time significantly slower than the Base GP.

A custom tournament selection, named as Partial Tournament, is used to reduce the number of repeated individuals in the crossover. Being P the population, the Partial Tournament does $|P|$ traditional K size tournaments, but limits the repeated number of each individual to two. Then, if necessary, the population is completed with individuals chosen at random up to $|P|$.

The GP settings used is presented in Table 4. The simplification rules used in RBS are as described in Table 1. The simple subtrees set S_{simple} used in EDS is the terminal set τ for each problem. The warm-up step used in DORE runs is defined as 200 full trees with depth limited in $[1, 5]$ range.

Results and Discussion The median was preferred over the mean in this section since it is less sensitive to outliers and the data is not guaranteed to follow a normal distribution. In all of the data in any graph or table the median of the 50 executions is shown.

Number of runs	50
Generations per run	50
Initialisation	Ramped half-and-half
Population size	200
Selection	Non-elitist Partial Tournament Selection, with $K = 5$
Crossover	*One Point Crossover* with 90% probability
Mutation	Uniform subtree mutation with 5% probability
Tree depth limit	Initial limit = 6; Subsequent limit = 17
Reduction threshold ρ	40%
Reduction threshold γ	60%

Table 4. GP settings used in symbolic regression benchmarks

It was first analyzed how the fitness behaved over generations. Figure 4 shows how Base GP is worse than the others in all benchmarks. It also presents how Base GP fitness curve tends to flatten in each benchmark. DORE has slightly better results than the baseline operator. Figure 5 shows a fitness boxplot of the best individuals in each generation.

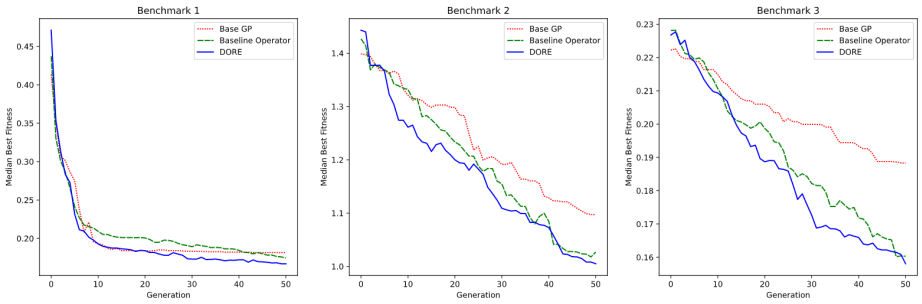

Fig. 4. Best fitness versus generations, for all benchmark problems

The size of individuals is a crucial concern to have bloat-controlled GP executions, beside the fitness stagnation. A tree depth limit was used, so it was not expected a completely uncontrolled growth on Base GP. As Figure 6 shows, both operators had a much better size control on the population over generations than Base GP. The difference between DORE and baseline operator was tiny, with DORE having best results on the first two benchmarks.

Table 5 shows the validation and test domains used to analyze the best individuals from each generation for all benchmark problems. It is presented in Table 6 the median and median average deviation (MAD), as utilized in [1], of

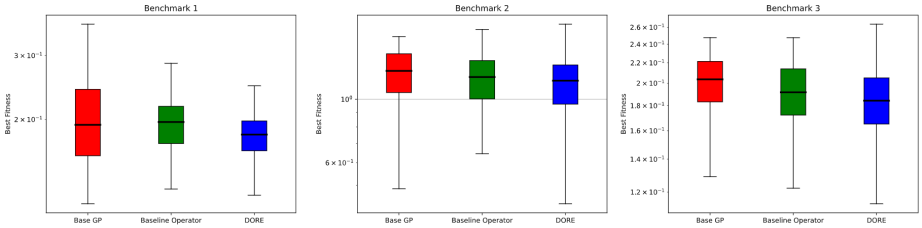

Fig. 5. Boxplots of the best of generation individuals, for all benchmark problems

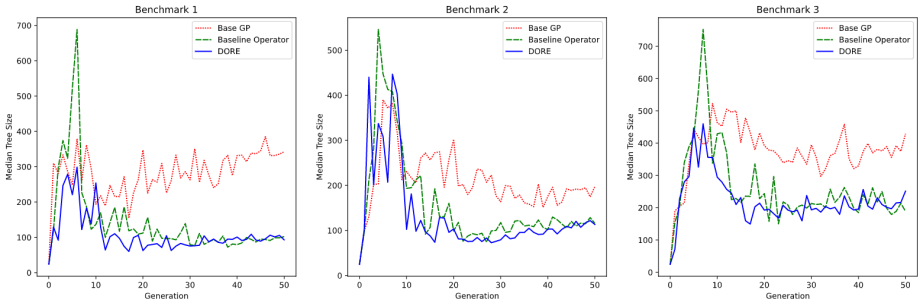

Fig. 6. Tree median size versus generation

each set of points - training, validation, and test - using the RMSE as reference. Table 7 shows the same metrics for the tree size of these best individuals. As the data is not normally distributed, Table 6 and Table 7 also present the p-value of the Mann-Whitney U-test [3] considering DORE x Base GP and DORE x Baseline Operator datasets. The null hypothesis is that the distributions of both datasets are equal. DORE performed better regarding tree size in all benchmark problems, compared to the other two strategies. It also presented better results in training error in all benchmarks. The benchmark 3 test error was worse using an operator, which suggests that this kind of strategy is not well suited for more complex problems even though the generated trees were smaller.

Benchmark	**Validation Domain**	**Test Domain**
1	100 random points $x_i \in [0, 6]$	500 random points $x_i \in [0, 6]$
2	100 random points $x_i \in [0, 6]$	1156 points $x_1, x_2 \in (-0,25 : 0,2 : 6,35)$
3	50 random points $x \in [-2, 2]$	2000 points $x \in (-2 : 0,001 : 2)$

Table 5. Validation and test domain for best of run individuals analysis

Method		Benchmark 1		Benchmark 2		Benchmark 3	
		Median	MAD	Median	MAD	Median	MAD
DORE	Training Error	0.18	1.77e-02	1.16	1.71e-01	0.18	1.95e-02
	Validation Error	0.22	2.46e-02	1.62	3.94e-01	0.26	4.58e-02
	Test Error	0.21	1.48e-02	1.09	4.49e-01	0.67	4.39e-01
Base GP	Training Error	0.19	4.03e-02	1.26	1.93e-01	0.20	1.79e-02
	(p-value)	(1.42e-16)		(1.49e-34)		(5.80e-78)	
	Validation Error	0.21	2.55e-02	1.75	4.02e-01	0.24	1.96e-02
	(p-value)	(1.28e-12)		(1.73e-27)		(5.23e-23)	
	Test Error	0.21	1.12e-02	1.59	8.89e-01	0.25	9.80e-03
	(p-value)	(4.12e-11)		(7.96e-09)		(3.78e-96)	
Baseline Operator	Training Error	0.20	2.38e-02	1.20	1.79e-01	0.19	2.10e-02
	(p-value)	(2.09e-34)		(1.19e-07)		(1.10e-16)	
	Validation Error	0.22	2.25e-02	1.62	3.72e-01	0.25	2.85e-02
	(p-value)	(5.81e-05)		(1.16e-01)		(3.94e-14)	
	Test Error	0.21	1.36e-02	1.24	7.44e-01	0.33	1.07e-01
	(p-value)	(1.84e-04)		(5.31e-02)		(6.43e-32)	

Table 6. Best individuals for generation performance, of all benchmark problems

Besides the best trees found for each generation, the best individuals of each run were also analyzed. Figure 7 shows the mean training fitness against the mean tree sizes. It is clear that DORE dominates the other strategies in all 3 benchmark problems regarding the fitness.

In benchmark 1, due to bloat, Base GP generates smaller but unfit individuals. These individuals are, in average, from earlier generations than the ones using code editing operators, see Figure 8. The boxplot shows how Base GP converges earlier than the other two strategies. Thus, it is expected that trees generated in later generations have better fitness but also a larger size. DORE performed better than the other strategies in Benchmarks 2 and 3 and regarding

Benchmark	DORE		Base GP			Baseline Operator		
	Median of Avg. Tree Size	MAD	Median of Avg. Tree Size	MAD	p-value	Median of Avg. Tree Size	MAD	p-value
1	94.55	6.11e+01	282.19	2.01e+02	5.61e-128	103.26	7.04e+01	1.37e-03
2	101.88	5.55e+01	196.58	1.21e+02	2.09e-87	119.84	7.96e+01	4.53e-10
3	203.80	1.34e+02	367.09	2.51e+02	8.29e-62	229.19	1.61e+02	1.30e-04

Table 7. Tree Size data of best individuals for generation, of all benchmark problems

both size and fitness, as shown in Table 6 and Table 7. In all benchmarks it is noted that the average size of the best individuals in DORE are better than the ones with the baseline operator, see Table 7.

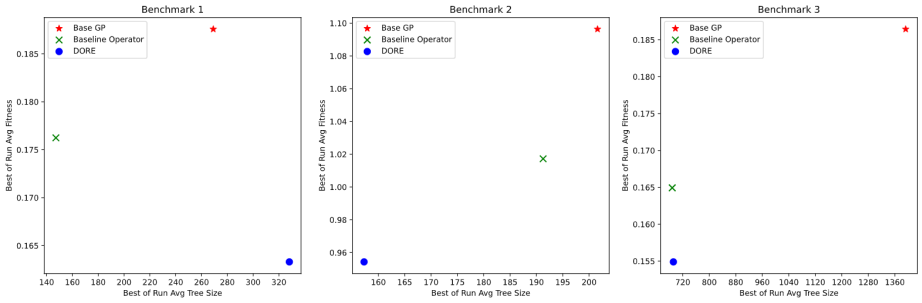

Fig. 7. Best of run average fitness against best of run average size

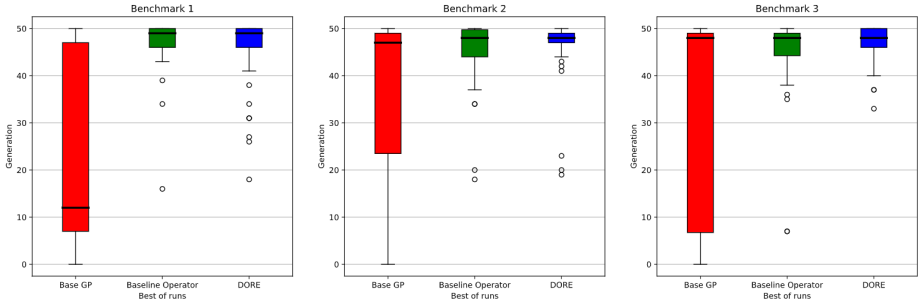

Fig. 8. Best of run generation boxplot

Another point of interest analyzed was the reductions made between baseline operator and DORE. Each RBS and EDS simplification made was logged in each run in these strategies. The tables and graphs related to reductions used the arithmetic mean as an average value. Table 8 highlights how DORE increased the number of unique simplifications made by RBS. This result was expected since DORE tends to increase the reductions table used by RBS.

The average value of total reductions made by RBS and EDS is shown in Table 9. These graphs present different scenarios. In benchmark 1, the total reductions made by both RBS and EDS in DORE was greater than in baseline operator. In benchmark 2, the reductions by RBS were greater in DORE but lesser by EDS. Finally, in benchmark 3, the total reductions made by both RBS and EDS was greater in the baseline operator. These different scenarios

Method	Benchmark 1	Benchmark 2	Benchmark 3
Baseline Operator	866.34	756.1	1735.98
DORE	1667.04	1330.9	2066.62

Table 8. Average value of unique RBS simplifications performed

Method	Avg. Total RBS simplifications			Avg. Total EDS simplifications		
	Benchmark 1	Benchmark 2	Benchmark 3	Benchmark 1	Benchmark 2	Benchmark 3
Baseline Op.	8060.32	12569.58	67622.52	104.8	165.92	680.82
DORE	16100.26	23830.96	39455.28	145.28	137.94	214.92

Table 9. Average total simplifications made by RBS and EDS

are directly related to the nature of each benchmark problem. Also, although the RBS reduction table in DORE had more entries than the baseline table, each simplification called by EDS triggered another recursive RBS simplification attempt on each subtree in the individual. Thus, in problems where the number of simplifications by EDS is high - and with an increased average size reduction - the number of reductions by RBS tends to be higher as well.

Even though each benchmark presented a different reduction scenario, a pattern in the ratio between reductions by RBS and reductions by EDS could be noted, as is shown in Figure 9. In all benchmark problems, except the proportion, the total amount of RBS reductions relative to the total amount of EDS reductions is bigger. It can also be noted that this ratio is greater in DORE than in the baseline operator, which is exactly the main goal of the proposed improvements.

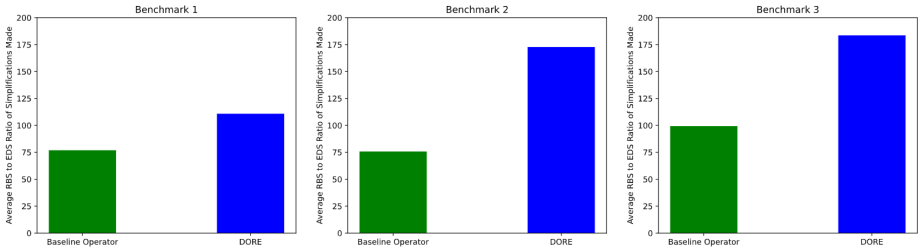

Fig. 9. Average ratio of simplifications made

The impact of the dynamically learned rules was analyzed and shown in Figure 10. The graphs are histograms of the total amount of reductions made by RBS in each subtree size reduction percentage interval. The values highlighted in purple are from rules that could only be learned with DORE improvements. The average size reduction of these dynamically learned rules in each benchmark were 86.01%, 88.01% and 85.40%, respectively, compared to the average size

reduction of 59.38%, 55.98% and 52.74% from the predefined rules. DORE not only provided more reductions to be applied but also better ones.

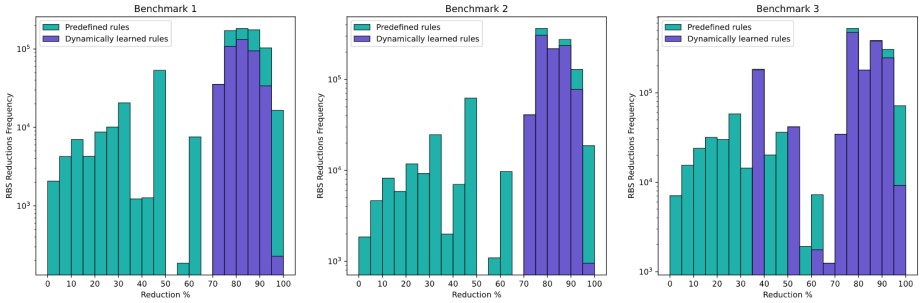

Fig. 10. RBS simplification frequency against size reduction percentage

Finally the execution time was analyzed, an important point of interest in this work as DORE execution time should be close to the baseline operator. Average execution time - using the arithmetic mean - for all benchmark problems is shown in Table 10. Since the values of parameters ρ and γ were defined empirically to make execution time between baseline operator and DORE close, the values presented are expected. However, the difference between Base GP and executions with operators is notable. Using RBS and EDS operators as a single bloat control method seems to be not adequate in hard or complex problems as the execution time tends to increase even more due to all the fitness function evaluation performed recursively in EDS step of the simplification flow.

Benchmark	**Average Execution Time (s)**		
	Base GP	**Baseline Operator**	**DORE**
1	180.36	292.23	321.12
2	167.31	276.08	277.59
3	206.10	585.60	514.68

Table 10. Average total execution time (in seconds) of each strategy, for all benchmark problems

6 Conclusion and Future Work

This paper presented improvements to a simplification flow introduced in [11] to make it more reasonable to be applied in complex problems. Besides a meta-learning reduction, two parameters were introduced to optimize the number of

trees to be reduced by RBS and EDS in harder problems than the one presented in the original work, without greater impact in execution time.

The resulting operator was able to improve the execution time performance of reduction by RBS, as well as the relative frequency of this reduction in the simplification flow, even with a limit of trees applied. A greater percentage of the population could pass through reduction by RBS with no significant penalty in execution time.

The results of the three artificial SR problems benchmark highlighted how DORE improved the fitness and slightly the tree size in the population during the evolutionary process. The best individuals in these experimental runs had significantly better results in both fitness and tree size using the dynamic operator.

It is noted that DORE is helpful to control bloat in GP but could be challenging to adopt in more complex problems due the computational burden. This issue should be addressed in future research. Probably tuning the parameters ρ and γ may be needed as well as limiting the size of the subtrees in S_{simple}.

Future work could measure the limits of the list S_{simple} and a robust strategy to handle random ephemeral constants, which was not explored in this paper. Parallelization of the reductions could improve the operator performance, making it more viable, although is not a trivial problem as the rule table is constantly updated by each EDS reduction performed. It would also be interesting to apply this operator with other bloat control methods, such as a equalization operator [14], using it as a tool to optimize a small set of individuals in a population.

References

1. Castelli, M., Manzoni, L., Mariot, L., Saletta, M.: Extending Local Search in Geometric Semantic Genetic Programming, pp. 775–787 (08 2019). https://doi.org/10.1007/978-3-030-30241-2_64
2. Chen, C., Luo, C., Jiang, Z.: Block building programming for symbolic regression. Neurocomputing **275**, 1973–1980 (2018). https://doi.org/https://doi.org/10.1016/j.neucom.2017.10.047
3. Fay, M.P., Proschan, M.A.: Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules. Statistics surveys **4**, 1 (2010)
4. Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**, 2171–2175 (jul 2012)
5. Haeri, M.A., Ebadzadeh, M.M., Folino, G.: Statistical genetic programming for symbolic regression. Applied Soft Computing **60**, 447–469 (2017)
6. Hagiwara, M.: Pseudo-hill climbing genetic algorithm (phga) for function optimization. In: Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan). vol. 1, pp. 713–716 vol.1 (1993). https://doi.org/10.1109/IJCNN.1993.714013
7. Hooper, D.C., Flann, N.S.: Improving the accuracy and robustness of genetic programming through expression simplification. In: Proceedings of the 1st Annual Conference on Genetic Programming. p. 428. MIT Press, Cambridge, MA, USA (1996)

8. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: European Conference on Genetic Programming. pp. 70–82. Springer (2003)
9. Koza, J.R., Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection, vol. 1. MIT press (1992)
10. McDermott, J., White, D.R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaskowski, W., Krawiec, K., Harper, R., De Jong, K., et al.: Genetic programming needs better benchmarks. In: Proceedings of the 14th annual conference on Genetic and evolutionary computation. pp. 791–798 (2012)
11. Mori, N., McKay, R., Hoai, N., Essam, D., Takeuchi, S.: A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification. vol. 13, pp. 237–244 (06 2009). https://doi.org/10i0079783642024818_24
12. Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Paul, W., Jordan, M.I., Stoica, I.: Ray: A distributed framework for emerging AI applications. CoRR (2017), http://arxiv.org/abs/1712.05889
13. Poli, R., McPhee, N.F.: Parsimony pressure made easy: Solving the problem of bloat in gp. In: Theory and Principled Methods for the Design of Metaheuristics, pp. 181–204. Springer (2014)
14. Silva, S., Dignum, S., Vanneschi, L.: Operator equalisation for bloat free genetic programming and a survey of bloat control methods. Genetic Programming and Evolvable Machines **13** (06 2012). https://doi.org/10.1007/s10710-011-9150-5
15. Sivanandam, S., Deepa, S.: Genetic Programming, pp. 131–163. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-73190-0_6, https://doi.org/10.1007/978-3-540-73190-0_6
16. Trujillo, L., Muñoz, L., Galván-López, E., Silva, S.: Neat genetic programming: Controlling bloat naturally. Information Sciences **333** (11 2015). https://doi.org/10.1016/j.ins.2015.11.010
17. Uy, N.Q., Hien, N.T., Hoai, N.X., O'Neill, M.: Improving the generalisation ability of genetic programming with semantic similarity based crossover. In: European Conference on Genetic Programming. pp. 184–195. Springer (2010)
18. Uy, N.Q., Hoai, N.X., O'Neill, M., McKay, R.I., Galván-López, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines **12**(2), 91–119 (2011)
19. Vladislavleva, E.J., Smits, G.F., Den Hertog, D.: Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Transactions on Evolutionary Computation **13**(2), 333–349 (2008)
20. Wong, P., Zhang, M.: Algebraic simplification of gp programs during evolution. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation. p. 927–934. GECCO '06, Association for Computing Machinery, New York, NY, USA (2006). https://doi.org/10.1145/1143997.1144156