Towards Automatic Grammatical Evolution for Real-world Symbolic Regression

Muhammad Sarmad Ali, Meghana Kshirsagar, Enrique Naredo and Conor Ryan
Biocomputing and Developmental Systems Lab, University of Limerick, Ireland

Keywords: Grammatical Evolution, Grammar Pruning, Effective Genome Length.

Abstract: AutoGE (Automatic Grammatical Evolution) is a tool designed to aid users of GE for the automatic estimation of Grammatical Evolution (GE) parameters, a key one being the grammar. The tool comprises of a rich suite of algorithms to assist in fine tuning a BNF (Backus-Naur Form) grammar to make it adaptable across a wide range of problems. It primarily facilitates the identification of better grammar structures and the choice of function sets to enhance existing fitness scores at a lower computational overhead. This research work discusses and reports experimental results for our *Production Rule Pruning algorithm* from AutoGE which employs a simple frequency-based approach for eliminating less useful productions. It captures the relationship between production rules and function sets involved in the problem domain to identify better grammar. The experimental study incorporates an extended function set and common grammar structures for grammar definition. Preliminary results based on ten popular real-world regression datasets demonstrate that the proposed algorithm not only identifies suitable grammar structures, but also prunes the grammar which results in shorter genome length for every problem, thus optimizing memory usage. Despite utilizing a fraction of budget in pruning, AutoGE was able to significantly enhance test scores for 3 problems.

1 INTRODUCTION

Grammatical Evolution (GE), since its inception twenty years ago, has found wide acceptance in the research communities (Ryan et al., 2018). It is a bio-inspired population-based methodology from the domain of evolutionary computation which heavily relies on the core aspect for its implementation: the definition of context-free grammar (CFG). By defining grammars in any language of choice, GE can evolve valid program of arbitrary length. This flexibility makes GE a powerful tool in genetic programming (GP) and it has gained a wide-scale appeal.

Grammar is a key input to grammatical evolution and it has been known that the performance of GE is significantly influenced by the design and structure of the grammar (Nicolau and Agapitos, 2018). However, when it comes to defining grammar, there is little guidance in the literature. This task is generally performed by the users of GE, solutions developers, or domain experts and the grammar is hand-crafted. Choice of terminals and non-terminals, and their composition to form production rules is largely based on expertise. For a novice user, there is no tool or framework which can assist them in defining the grammar.

A related problem, faced even by the experienced users, is that of the choice of function set. Functions or operators are represented as productions in the grammar. Choosing an appropriate function set is a key decision in applying GP as it can have a vital impact on the performance of GP (Gang and Soule, 2004; Uy et al., 2013). However, there is not enough guidance in selecting a function set and no systematic approach exists (Nicolau and Agapitos, 2021). To date, it is also largely considered a decision made by domain experts.

Automatic Grammatical Evolution (AutoGE) (Ali et al., 2021) is a system that can aid users of GE to explore and identify grammar structures to smoothly adapt according to the underlying problem domain. It can aid users in identifying appropriate terminals involved in forming production rules. It is being developed with a rich suite of algorithms which can adapt (prune or extend) user provided grammar, or even generate an appropriate grammar from scratch if certain pieces of information about problem at hand are known. Besides definition and/or fine-tuning of the grammar, AutoGE will facilitate in adapting other evolutionary parameters such as mutation/crossover probabilities and tree depths. Depending upon the

Ali, M., Kshirsagar, M., Naredo, E. and Ryan, C.
Towards Automatic Grammatical Evolution for Real-world Symbolic Regression.
DOI: 10.5220/0010691500003063
In *Proceedings of the 13th International Joint Conference on Computational Intelligence (IJCCI 2021)*, pages 68-78
ISBN: 978-989-758-534-0; ISSN: 2184-3236

nature of the problem and its complexity, it can assist in the selection and definition of correct fitness function, which can be composed of single, multiple or many objectives, and can be hierarchical in nature (Ryan et al., 2020).

This work reports preliminary results with our Production Rule Pruning approach applied to real-world symbolic regression problems. For a given grammar structure and a generic larger function set, it reduces the grammar by pruning useless productions. It helps in evolving individuals of shorter lengths thereby optimizing memory usage (Kshirsagar et al., 2020). The algorithm and related production ranking scheme is discussed in section 4. Section 5 shows our experimental setup and section 6 presents and discusses the results. In the coming section 2 and 3, we briefly outline theoretical background and the related work.

2 BACKGROUND

2.1 Grammatical Evolution

Grammatical Evolution is a variant of Genetic Programming (GP) in which the space of possible solutions is specified through a grammar. Although different types of grammars have been used (Ortega et al., 2007; Patten and Ryan, 2015), the most commonly used is Context Free Grammar (CFG), generally written in Backus-Naur Form (BNF). GE facilitates a modular design, which means that any search engine can be used, although typically a variable-length Genetic Algorithm (GA) is employed to evolve a population of binary strings.

In GE, each population individual has a dual representation, a genotype and a phenotype. When the underlying search engine is a genetic algorithm, the genotype is a sequence of codons (usually a group of 8-bit substrings), while the phenotype expresses an individual's representation in the solution space. Mapping, a key process in GE, maps a given genotype to the phenotype. While subsequently consuming each codon, it selects a production from the available set of alternative productions in a rule through mod operations and builds the derivation tree (Ryan et al., 1998). Although there are other mapping schemes (Fagan and Murphy, 2018), the conventional scheme follows left-most derivation. An important measure in the mapping process is the *effective genome length*, which is equal to the number of codons consumed to generate a fully mapped individual (the one which does not contain any non-terminals in its phenotype). The actual genome length is the total number of

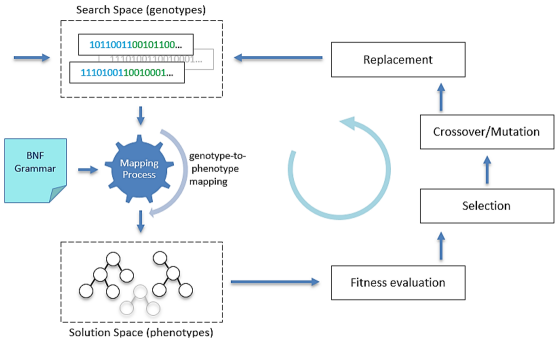

Figure 1: Schematic of Evolutionary Process in GE.

codons in the genome, some of which may remain unused.

2.2 Grammar Design

Since GE exploits the expressive power of grammars, it can be applied to a multitude of problem domains, for instance in Symbolic Regression (SR) where the purpose is to search the space of mathematical expressions to find a model that best fits a given dataset (Koza, 1993). To construct valid and useful mathematical expressions in GE, the grammar needs to be well designed.

A grammar is formally defined as the tuple (T, N, P, S) where T is a set of terminal symbols, N is a set of non-terminal symbols, P is a set of production rules, and S is the start symbol . While the set of terminals outline the building blocks of a solution, the choice of non-terminals and deciding how exactly to organize those into a set of rules and productions is a design task. By designing an 'appropriate' grammar, one specifies the syntactic space of possible solutions (it is worth noting that there are an infinite number of possible grammars which specify the same syntax). Although grammar design is an important consideration, yet majority of the research works provide little to no justification for the design decisions related to the choice of (non)terminals and the formation of production rules.

2.3 Grammar Structures

Instead of designing grammar from scratch, a common approach is to utilize and adapt existing grammar designs for that domain. For example, in grammatical evolution based symbolic regression (GESR), typical grammar structures are shown in Table 1. The most widely used structure, which we call *mixed-arity* grammar, combines operations of multiple arities in a single rule. A contrasting structure is that of *arity-*

Table 1: Grammar Structures.

Structure	Rules
Mixed-arity Grammar	<expr> ::= <expr><op><expr> |sin(<expr>)|cos(<expr>) |exp(<expr>)|pow(<expr>,2) |sqrt(<expr>)|<var> <op> ::= + | - | * | / <var> ::= X | Y
Arity-based Grammar	<expr> ::= <expr1>|<expr2>|<var> <expr1> ::= sin(<expr>)|cos(<expr>) |exp(<expr>)|pow(<expr>,2) |sqrt(<expr>) <expr2> ::= <expr>+<expr>|<expr>-<expr> |<expr>*<expr>|<expr>/<expr> <var> ::= X | Y
Balanced Arity-based Grammar	<expr> ::= <expr1>|<var> |<expr2>|<var> ... # The rest is the same as arity-based

based grammars where productions relevant to arity-1 and arity-2 operations are grouped in separate rules. A *balanced* grammar version balances the probabilities of selecting recursive (non-terminating) productions and terminating productions (Nicolau and Agapitos, 2018).

It is important to note how operators and functions are represented as productions in the grammar. Besides embodying arithmetic operators, a number of common mathematical functions are represented as alternative recursive productions.

3 RELATED WORK

In this work, we exercised our approach on problems in symbolic regression which is the most common application domain for GP-like systems. Since there is a large amount of work in relation to symbolic regression, we skip that discussion due to space limitation and rather focus on the following relevant research directions:

3.1 Function Set Selection

It is important to select appropriate function set in order to achieve good performance in GE/GP. Not many works appeared which specifically address the problem of function set selection. (Gang and Soule, 2004) experimented with various function sets and highlighted that function groups exist and functions in the same group have same effect on performance. (Uy et al., 2013) examined characteristics of the fitness landscapes generated by various function sets and the performance of GP. They concluded that the autocorrelation function can be used as an indicator to select a function set. Recently, (Nicolau and Agapitos, 2021) also studied the effect of various groups of function sets on generalisation performance of GP and GE. With a detailed review and experimentation over a large set of symbolic regression problems, they concluded that protected functions should be avoided. They also indicate that full set (comprising of all considered function primitives) performed consistently well in training across all problems. Our earlier study (Ali et al., 2021) support their finding, while we use a larger generic function set at the start of evolutionary process.

3.2 Encapsulation

In a normal setup of canonical GE, grammar is a static artefact which never changes during the execution. However, in this work, we modify the grammar by removing productions from the grammar which we term as *pruning*. Several strands of research modify the grammar dynamically during the evolutionary process. The idea of automatically defined functions, instead of striving to choose an optimal set in advance, is about identifying, encapsulating, and reusing useful functionality discovered during the evolution (Koza, 1994). (O'Neill and Ryan, 2000) used grammar-based approach to automatically define new function for the Santa Fe trail problem. (Harper and Blair, 2006) introduced a meta-grammar into grammatical evolution allowing the grammar to dynamically define functions without the need for special purpose operators or constraints. More recently, (Murphy and Ryan, 2020) utilized covariance between traits to identify useful modules which are added to the grammar.

3.3 Probabilistic GE

In our work, we assign weight called *rank* to a production, which at a stage is used to decide upon its fate: whether or not to stay in the grammar. Although our ranks do not bias the selection of a production during the mapping process, the probabilistic approach to GE does . It uses probabilistic grammar, also known as stochastic context-free grammar (SCFG) to assign selection probabilities to each production. Although a huge set of research explores probabilistic grammars in connection to GP and Estimation of Distribution Algorithms (EDA), there aren't attempts to utilize SCFG in GE with genetic operations, except the recent work from (Megane et al., 2021). This paper does not compare our ranking approach with SCFG, which is a definite future work.

4 METHODOLOGY

We discuss our approach to rank grammar productions and subsequent pruning of unworthy productions in this section. Prior to that, we present our hypothesis underlying this approach.

4.1 Hypothesis

It is well known that with the correct configuration and fitness criteria, an evolutionary process is geared towards convergence. Increasingly, the evolved solutions contain more and more of the right ingredients or building blocks (in our case, grammar productions) (Koza, 1993). We hypothesize that the structural composition of evolved solutions carries information that can be useful in identifying the right ingredients.

In GE, each individual in the population is composed of terminals, which appear in an order defined by the derivation tree constructed during genotype to phenotype mapping. By traversing the derivation tree, it is possible to obtain a list of grammar productions used in the mapping process to generate an individual. Such a list is termed as the *production-list*. Once identified, the frequency of usage of each production in the production-list can be easily determined.

Productions can be weighed or ranked based on how frequently they are used in the construction of individuals in the population. As evolution proceeds, fitter individuals survive, and the productions which more frequently shape the structures are the ones that are considered to be worthy being part of the grammar. Such productions should be assigned a high rank. Conversely, productions which harm individual's fitness to an extent they become extinct, generally do not enjoy high usage frequency (although rarely zero, due to hitch-hiking effects) in the population.

To test our hypothesis, we devised a simple frequency-based approach to rank productions, which we present next.

4.2 Production Ranking

Figure 2 describes the overall process of production ranking. At the end of a generation, individuals in the population are structurally analyzed and assigned ranking scores based on the frequency count of productions in the production-list. requency ofjth production

Let P be the set of productions in the grammar G. $P_i \subset P$ is the set of productions in the production-list of the ith individual. If $n = |P|$, number of productions in the P, and $k = |P_i|$, then k < n for practically

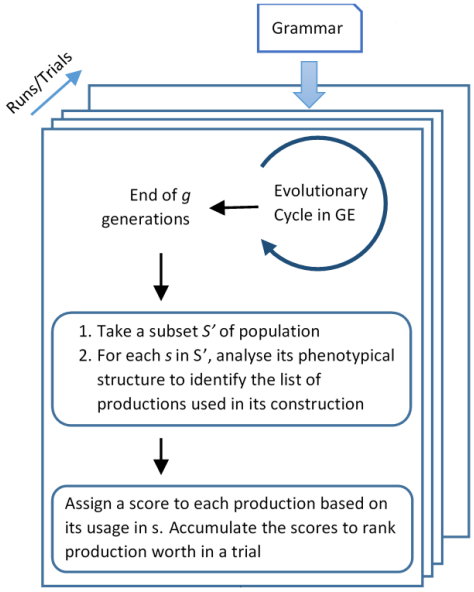

Figure 2: Schematic of production ranking at each Stage.

all individuals. The ranking score assigned to the jth production in the production-list is given by:

$$(nfr)_i^j = \left(\frac{\phi_i^j}{l_i} \right) \quad (1)$$

$$(fpr)_i^j = (nfr)_i^j \times \rho_i \quad (2)$$

where ϕ_i^j is the frequency of jth production, l_i is the effective codon length, and ρ_i is the fitness of ith individual. Equation 1 defines the *normalized frequency rank* (*nfr*) of a production, while Equation 2 computes the *fitness-proportionate rank* (*fpr*). As a consequence of the above two definitions, the following two properties hold for an ith individual:

$$\sum_{j=1}^{k} (nfr)_i^j = 1 \quad \text{, and} \quad \sum_{j=1}^{k} (fpr)_i^j = \rho_i$$

Once individual ranking scores have been computed, we accumulate the scores of all u individuals in the population to compute *generation worth* (*gw*) of jth production in the production-list for mth generation, and then across all g generations to compute the overall *run worth* (*rw*).

$$(gw)_m^j = \sum_{i=1}^{u} (fpr)_i^j \quad (3)$$

$$(rw)^j = \sum_{m=1}^{g} (gw)_m^j \quad (4)$$

To minimize the computational cost of production ranking, we track the production-list during the mapping process, so it does incur a small memory overhead. However, since the ranking scores are computed at the end of a small number of evolutionary

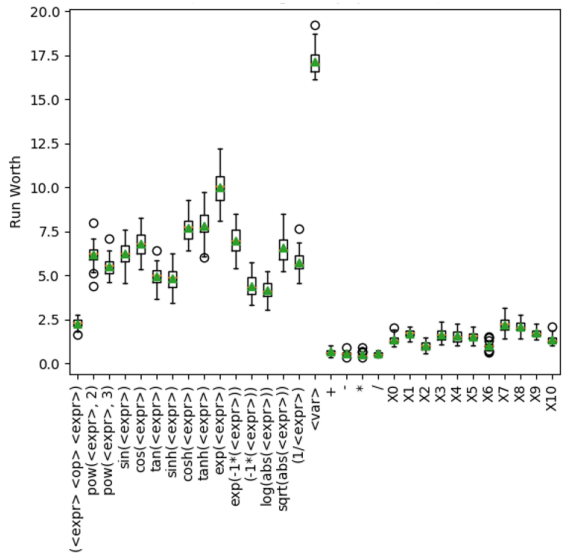

Figure 3: Fitness-proportionate production ranks across runs for `redwine` dataset (runs: 30, ngen: 5, popsize 250).

runs (called a *stage*), and the operations defined by the above equations are trivial, those can be efficiently performed with minimal overhead.

An important consideration is to decide how much of the population to select for ranking. We experimented with three possible choices: 1) the whole population, 2) only unique individuals, 3) top X% of the population (we use X=20). The second option turned out to be the best choice based on our empirical evaluations. Potential issue with the first option is that the rankings can be biased due to redundancy, and with the third option there is a chance of pruning important production which is not yet picked because of the small number of evolutionary iterations.

Figure 3 shows a sample box and whisker plot of *fpr* ranking for the `redwine` dataset. It gives a nice picture of the utility of each production in the evolutionary cycle.

4.3 Grammar Pruning

According to Occam's razor, "no more things should be presumed to exist than are absolutely necessary." Following this principle, we try limiting the complexity of the models and favour simpler ones to take part in the evolution. Grammar is a key model of the solution space, so the idea is to remove unnecessary or less worthy productions (or functions) from the grammar to *tune* the grammar design.

One of the key driver in grammar tuning[1] is the pruning strategy and algorithm. There can be a number of strategies for pruning and we look at two here. The core idea they have in common is a staged approach; that is, at each stage solutions are evolved over a small number of generations, then one or more productions are pruned, and then subsequent stages (if any) are conducted. Every next stage is a complete restart with the newly modified grammar (more on this in section 6.3). The number of stages may vary depending upon the strategy being employed. We experimented with the following two strategies:

- Strategy 1: Prune for the maximum pruning budget (20%). The remaining runs will verify if it was fruitful.
- Strategy 2: Only proceed with pruning if it results in improving mean training score at each stage. If it degrades performance, stop.

Strategy 1 has slightly less overhead as it is composed of only one stage, but suffers from blind pruning which in many cases fails to reap any benefits. Strategy 2 incorporates a feedback loop which informs on its usefulness. In our preliminary experiments, we have observed it to be yielding a much better overall outcome. Coupled with the pruning policies defined in section 5.4, our Production Rule Pruning algorithm achieved good results.

The pseudocode listed in Algorithm 1 outlines Production Rule Pruning (PRP) algorithm. Choice of pruning budget B_{gt} and number of generations for pruning runs gen_p determine the maximum possible stages. In our experimentation, We set B_{gt} to 20 and gen_p as 5. `PRUNE` is the key procedure in the algorithm. It performs two important functions:

Algorithm 1: Production Rule Pruning (PRP).

input : grammar G, number of trials/runs T, available budget B_{gt}, number of generations for pruning runs gen_p
output: pruned grammar G_p

```
initialize MB_prev to a high value;
G_p ← G;
while B_gt > gen_p do
    do T runs for gen_p with grammar G_p;
    MB_curr ← get current mean-best;
    S_prune ← PRUNABLES();
    decrement B_gt;
    if MB_curr < MB_prev then
        G_p ← PRUNE();
        MB_prev ← MB_curr;
    else
        REVERT();
        increment gen_p;
    end
end
```

[1]It is worth mentioning that in AutoGE, tuning may involve pruning as well as extension of the grammar, although this work only reports on pruning approach.

1. It analyses production ranking scores and identifies the least worthy productions. Based on the pruning policy, it identifies how many productions to prune at a given stage and returns that many productions as candidates to be pruned.
2. It removes productions from the grammar and adds them to S_{prune} which is implemented as a stack. At each stage, pruned productions are pushed to the stack.

The `REVERT` function undoes the last pruning action by popping the last productions from S_{prune} and adding them back to the grammar. When a pruning stage reverts and the budget is still remaining, gen_p is incremented, in our case from 5 to 10.

The output of the `PRUNABLE` function are pruning suggestions. In our earlier work (Ali et al., 2021), we empirically evaluated the consistency of first and second pruning suggestions (we only prune 2 productions at max in a stage) and found those to be 99% and 95% consistent respectively over 100 experimental runs for certain randomly chosen problems.

5 EXPERIMENTAL SETUP

5.1 Dataset

Table 2 lists the problems considered in this work. All the datasets correspond to the real-world symbolic regression benchmark problems which have been widely studied in several esteemed publication venues, as also noticed by (Oliveira et al., 2018; Raymond et al., 2020). Except for Dow Chemical dataset, which was sourced from gpbenchmarks.org website[2], all other datasets were obtained from the UCI Machine Learning Repository[3] and CMU StatLib Archive[4].

The collection of datasets is diverse including problems having 5 to 57 input features, with sample size varying from 60 to nearly 4900. There are no missing values in the dataset, and we utilize the raw values without any normalization. Each dataset is referred to with a short name (in distinct font), which will be used in the rest of the paper.

5.2 Parameters

Table 3 presents the evolutionary parameters used in all experimental runs. Note that we utilized *repeated* k-fold cross validation, with k = 10 and repeat factor r = 3. For repetition we used a different seed to prepare a different training-test data split each time. The repetition ensure that we further minimize the chances of overfitting (Wong and Yeh, 2020).

Table 2: List of Datasets.

Dataset	Short name	Features	Instances
Airfoil Self-Noise	`airfoil`	5	1503
Energy Efficiency - Heating	`heating`	8	768
Energy Efficiency - Cooling	`cooling`	8	768
Concrete Strength	`concrete`	8	1030
Diabetes	`diabetes`	10	442
Wine Quality - Red Wine	`redwine`	11	1599
Wine Quality - White Wine	`whitewine`	11	4898
Boston Housing	`housing`	13	506
Air Pollution	`pollution`	15	60
Dow Chemical	`dowchem`	57	1066

Table 3: Parameter Settings.

Parameter	Value
Number of Runs	30
Population Size	250
Number of Generations	100
Search Engine	Steady-State GA
Cross Validation	10-fold (r=3)
Crossover Type	Effective Crossover
Crossover Probability	0.9
Mutation Probability	0.01
Selection Type	Tournament
Initialization Method	Sensible Initialization
Max Pruning Budget	20% (4 stages at max)
Extended function set	$+ \quad - \quad \times \quad / \quad -x \quad x^2 \quad x^3$ $sin \quad cos \quad tan \quad sinh \quad cosh$ $tanh \quad e^x \quad e^{-x} \quad ln\|x\| \quad \sqrt{\|x\|}$

We ran all experiments on the libGE system[5], which is an efficient C/C++ implementation of canonical GE and provides capabilities to effectively examine grammar productions.

The purpose of the fitness function is to measure the performance of the algorithm against a predefined objective. A common fitness function in symbolic regression is Root Mean Squared Error (RMSE), which is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

where n is the number of data points, y_i is the target value, and $\hat{y}_i$ is the predicted value. RMSE assesses the mean extent of deviation from the desired value, so the goal of evolution is to minimize this error metric across generations.

[2] http://gpbenchmarks.org/?page_id=30
[3] https://archive.ics.uci.edu/ml/datasets.php
[4] http://lib.stat.cmu.edu/datasets/
[5] http://bds.ul.ie/grammatical-evolution/

5.3 Grammars and Function Set

We defined an extended function set (see Table 3), which is the superset of all mathematical functions commonly used in symbolic regression. It includes arithmetic operators, trigonometric functions, exponential and power functions. We do not use protected division. However we did include functions such as e^x, e^{-x}, *tan*, *sinh*, and *cosh*. These functions grow exponentially and are usually avoided (Nicolau and Agapitos, 2021). However, we kept those in our function set in order to validate if our approach of production ranking and grammar pruning was able to remove such functions from the grammar.

The grammar which contains productions embodying whole extended function set is called *Extended Grammar* in this work (referenced with the letter 'E' in the results). The three grammar structures considered are shown in Table 1. The `<var>` rule includes as many alternative terminal productions as there are number of input variables in the dataset.

5.4 Pruning Policies

For the problems we examined, the usage frequency of arithmetic operators, input features, and constant terminals was low, irrespective of the grammar structure. We therefore do not consider their corresponding productions, and the productions where the right hand side is only composed of non-terminals (for example productions in the start rule of arity-based grammar in Table 1). This resulted in 14 prunable productions (excluding productions embodying arithmetic operators).

It is important to highlight a few other policies adopted while pruning which server as parameters to the PRP algorithm:

- We do not consume more than 20% of the computational budget on pruning. In our case, it meant consuming at most 20 generations;
- Pruning takes place in stages. At a stage, prune only 10% of the productions;
- In pruning runs, we evolve for 5 generations to maximize pruning. If 5 generation runs terminate with a positive `REVERT` decision and part of the pruning budget is remaining, we proceed with 10 generations.

Note that the grammar which results after pruning is termed *Pruned Grammar* in this work and is referenced with the letter 'P' in Table 4.

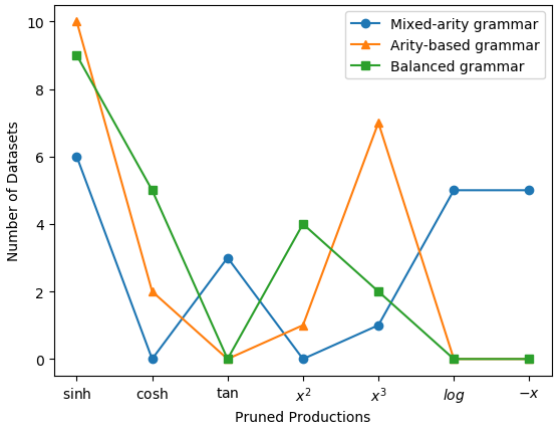

Figure 4: Productions pruned in first two stages of pruning across all datasets.

6 RESULTS

We conducted an extensive set of experiments over the 10 datasets. A single experiment comprised of 30 runs for each given problem and grammar structure. Table 4 presents a summarized view of the results. It shows the impact of using various grammar structures alongside extended function set and pruning approach on training performance, test performance, and mean effective genome length for the best-of-run solution. The letter on the right of each cell indicates which grammar ('E' for the grammar with extended function set, 'P' for pruned) achieved better results. Results in italics indicate which grammar structure scored the highest. When the numbers are in boldface, the differences are statistically significant. The cells in yellow indicate that pruning results in better scores, regardless of significance.

6.1 Statistical Comparisons

Since the assumption of normality and dependence, as required by parametric tests, does not hold in general in experimental results of evolutionary computing approaches, we decided to use non-parametric tests, for which we followed the guidelines presented by (Derrac et al., 2011). In our work, we do pairwise as well as multiple comparisons. All results were compared at the 0.05 statistical significance level.

For each problem, and each grammar structure, we compared the best training, test, and effective genome length scores (in 30 runs) among extended and pruned grammars using a non-parametric Mann-Whitney U-test (2-tailed version). Scores in boldface indicate that the p-value, while comparing outcomes resulting from extended vs. pruned grammar for a

Table 4: Effective genome length, test, and training performance comparisons. 'E' stands for extended, 'P' for pruned; numbers in italics indicate which grammar structure scored best; bold indicates statistical significance; yellow highlights indicate improvement due to pruning.

	Effective Size			Training			Test		
Dataset	**Mixed**	**Arity-based**	**Balanced**	**Mixed**	**Arity-based**	**Balanced**	**Mixed**	**Arity-based**	**Balanced**
redwine	***16.86 (12.23) P***	**17.06 (7.87) P**	23.6 (14.23) P	**0.723 (0.025) E**	0.721 (0.024) E	*0.706 (0.020) E*	0.742 (0.060) P	0.719 (0.058) E	***0.709 (0.067) P***
diabetes	***14.33 (6.21) P***	17.4 (3.729) P	**23.8 (10.33) P**	62.43 (3.735) E	59.77 (1.880) P	*59.67 (1.794) E*	63.71 (5.943) P	*59.88 (4.548) E*	**60.36 (4.001) P**
housing	***15.26 (9.19) P***	18.4 (7.292) P	28.06 (13.70) P	6.747 (0.514) P	**6.499 (0.573) E**	*6.437 (0.601) E*	6.737 (2.053) P	6.695 (1.597) E	*6.434 (1.716) E*
whitewine	17.93 (11.75) E	***16.0 (7.95) P***	**17.6 (7.012) P**	**0.826 (0.021) E**	**0.806 (0.015) P**	***0.799 (0.009) P***	**0.826 (0.029) E**	**0.804 (0.022) P**	*0.799 (0.022) P*
dowchem	***16.4 (11.09) P***	19.66 (8.051) E	21.33 (14.84) E	***0.336 (0.009) P***	0.337 (0.009) P	0.338 (0.009) P	*0.323 (0.031) E*	0.338 (0.034) E	0.338 (0.035) E
concrete	***19.8 (11.45) P***	22.53 (6.751) E	33.6 (20.92) P	**13.83 (0.878) E**	13.36 (1.014) E	*12.98 (1.310) P*	13.87 (1.281) E	13.61 (1.195) E	*13.05 (1.645) E*
pollution	*20.46 (12.42) P*	22.46 (8.143) E	**24.0 (11.78) P**	***55.65 (2.450) E***	**81.93 (11.65) E**	**79.45 (8.584) E**	*55.97 (18.32) E*	79.11 (26.39) E	84.22 (36.61) E
airfoil	*23.56 (16.23) P*	28.86 (9.175) E	44.93 (14.76) P	*6.673 (0.098) E*	**25.53 (5.608) E**	25.43 (7.329) P	*6.688 (0.319) E*	**25.63 (5.870) E**	25.53 (7.208) P
heating	*22.66 (18.21) P*	23.06 (8.242) E	29.06 (9.87) P	**4.368 (0.373) E**	**4.340 (0.187) E**	*4.132 (0.306) E*	**4.342 (0.515) E**	4.381 (0.356) E	*4.150 (0.562) E*
cooling	*16.06 (5.88) P*	25.13 (11.08) P	36.06 (17.95) E	**4.170 (0.172) E**	**4.044 (0.089) E**	*3.979 (0.243) E*	**4.169 (0.368) E**	4.049 (0.349) E	*3.935 (0.340) E*

fixed grammar structure, was less than 0.05 and the null hypothesis was therefore rejected.

To compare among three grammar structures (mixed, arity-based, balanced), we utilized Friedman test, where the null hypothesis assumes no effect or difference for adopting any of the grammar structures with respect to training performance, test performance or genome length. Where the null hypothesis was rejected, the post-hoc analysis was carried out using Shaffer's static procedure (Derrac et al., 2011) to signify which pair of grammars yielded statistically different results. The outcome is discussed in next section.

6.2 Effect of Grammar Structures

It is evident that some grammar structures are more appropriate for certain types of improvement or for certain problems. When comparing raw numbers from Table 4:

- Mixed-arity grammar results in lower genome length for all problems except `whitewine`, for which arity-based structure was the best choice achieving significantly better training as well as test performance;
- Balanced arity-based grammar achieve better results in 7 problems (in training) and 5 problems (in test) out of 10 problems;
- An interesting set of results appear for `pollution` and `airfoil` datasets where mixed-arity grammar achieved much better results as compared to other two structures. This trend was observable from very early on in the evolution.

For both extended and pruned grammars, we compared among three groups (named as, M for Mixed-arity, A for Arity-based, and B for Balanced arity-based) of results for each type of the three outcomes (training, test, and effective size) using Friedman test. For instance, considering extended grammar, we ran a test which compared mean training scores for all 10 problems in case of mixed-arity, arity-based, and balanced-arity grammar structures. In this way, we ran 6 Friedman tests, comparing among results of 3 grammar structures in each. In case of training and test groups, no significant difference was found. However, for the effective size group, the null hypothesis was rejected ($p < 0.05$) which indicates there are significant differences among the outcomes of three grammar structures.

For post-hoc analysis, we utilized Shaffer's static procedure, as recommended in (Derrac et al., 2011). With the pair-wise multiple comparisons among grammar structures, following two pairs were identified as producing significant difference:

- Mixed-arity vs. Arity-based for extended grammar with adjusted p-value (APV) of 0.02534;
- Mixed-arity vs. Balanced for pruned grammar with APV of 0.00346.

Since in both pair mixed-arity achieved better results, it was concluded that the mixed-arity structure significantly improves effective genome length with or without pruning. Many symbolic regression studies using GE or Grammar Guided Genetic Programming (GGGP) use mixed-arity grammar structure, for instance (Nicolau et al., 2015). Based on our findings, we state that the choice is fruitful in exploring small-sized less bloated solutions, but it does not warrant gains in generalization or approximation.

6.3 Effect of Grammar Pruning

Pruning was applied in all experiments. The number of productions pruned varied from 2 to 6, out of 14 prunable productions. Figure 4 shows for how many different problems a production was pruned in first two stages of pruning with different grammar structures. *sinh*, *cosh*, and x^3 were actively recognized

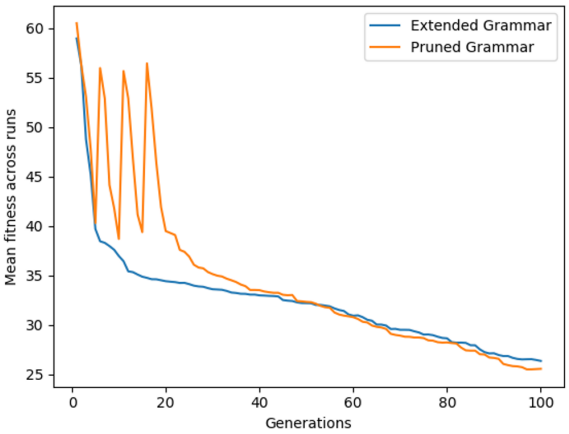

Figure 5: Impact of grammar pruning on test performance of *airfoil*.

as unuseful productions even in the initial generations. It would be interesting to explore why Mixed-arity grammars rank log and $-x$ so low. It is evident from the highlighted cells in Table 4 that pruning further improves/reduces genome length, especially with mixed-arity or balanced arity-based structures. Figure 6 shows the same effect in box plot for some of the problems.

Table 5 shows the percentage improvement with grammar pruning approach comparing extended grammar and pruned grammar results. Again, these differences are mostly significant in case of effective size evaluations. Besides, following inferences can be drawn from these results:

- Genome length improvements due to pruning are more prominent with mixed-arity grammar.
- Pruning resulted in improved test performance in `housing`, `diabetes`, `redwine`, and `airfoil` datasets, though gains are non-significant.
- For three datasets (`cooling`, `heating`, and `whitewine`), pruning, with the currently experimented strategy, could not improve test performance. Also, for the rest of the problems, the drop in test performance was not significant.

However, it is worth noting that since we keep the same computational budget, trials with the pruned grammar lasted for 80 (in some cases 85) generations. Had the pruned grammar also exercised for 100 generations, it is likely that it would have achieved better performance. Figure 5 shows a sample convergence plots (for `airfoil` problems) where grammar pruning enhanced generalization performance when compared with extended grammar. The three spikes in the plot in case of pruned grammar depict that pruning runs were carried out in three stages in the first 15 generations.

Table 5: Percentage Improvement with Pruning.

	Mixed-arity		Balanced	
Dataset	**Test**	**Eff. Size**	**Test**	**Eff. Size**
`housing`	4.32%	**39.73%**	-5.28%	2.09%
`diabetes`	1.09%	**37.04%**	0.94%	**22.05%**
`redwine`	0.08%	**29.33%**	0.20%	18.62%
`airfoil`	-0.53%	39.42%	3.13%	25.44%
`dowchem`	-1.56%	**30.90%**	-0.15%	-4.06%
`concrete`	-3.96%	**31.01%**	-0.45%	5.26%
`cooling`	**-4.53%**	39.37%	-4.69%	-3.88%
`heating`	**-8.29%**	39.77%	-2.16%	18.50%
`pollution`	-12.59%	19.74%	-6.81%	**28.43%**
`whitewine`	**-1.99%**	**-7.25%**	1.68%	**47.20%**

7 CONCLUSION

We propose a new algorithm as part of the AutoGE tool suite being developed. The proposed Production Rule Pruning algorithm is an approach that combines an extended function set and a frequency counting mechanism for ranking production rules. AutoGE achieved significantly better genome length in 7 out of 10 problems (with improvements the other three also), without significantly compromising on test performance of any, while in three of the problems, AutoGE shows a significant improvement on test performance. Our results highlighted that mixed-arity grammar structure or balanced arity-based structure can be a better choice for real-world symbolic regression problems.

7.1 Future Directions

An immediate extension to the current work is to improve and trial grammar pruning approach for feature selection and how it impacts test performance. Besides, dynamic approach to pruning and improved ranking schemes and pruning strategies are also planned. We also intend to explore other problem domains for instance program synthesis, and Boolean logic. The PRP algorithm performance can be further enhanced by investigating other search mechanisms, for example particle swarm optimization or ant colony optimization. We aim to extend AutoGE's suite of algorithms and to make it more robust by exploring approaches like grammar-based EDAs.

ACKNOWLEDGEMENTS

This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094_2 and co-funded under the European Regional Development Fund through the Southern & Eastern Re-

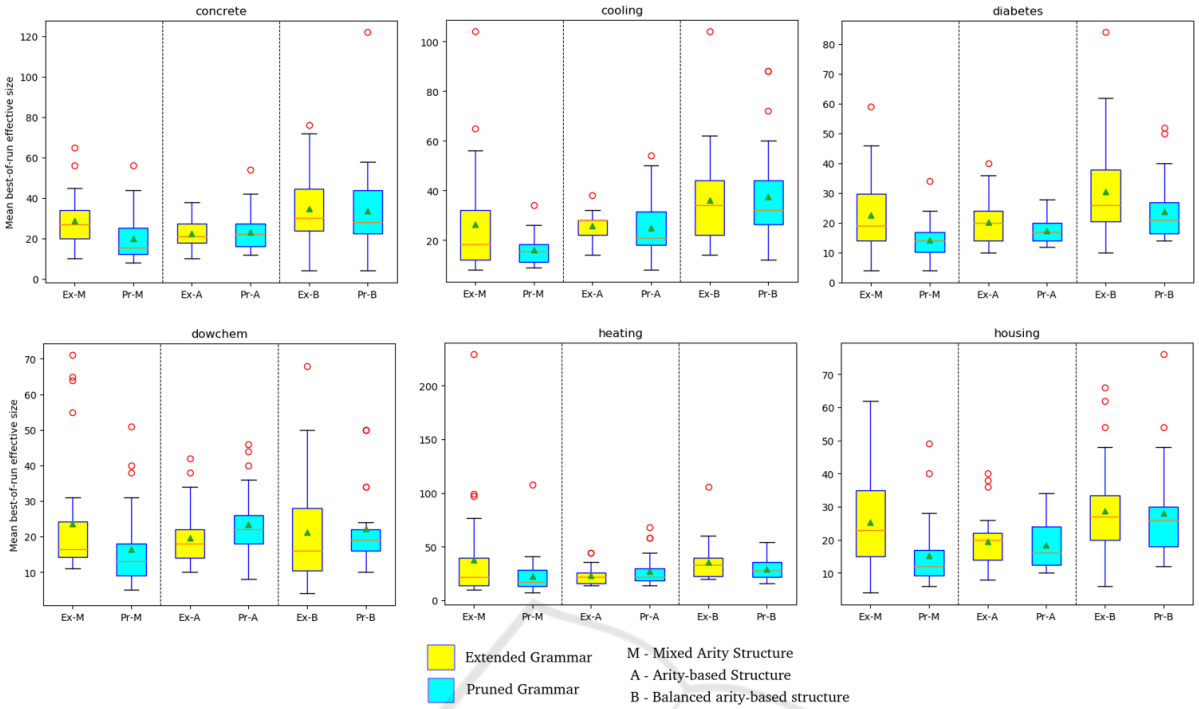

Figure 6: Impact of Grammar Pruning on Effective Size.

gional Operational Programme to Lero - the Science Foundation Ireland Research Centre for Software (www.lero.ie)

REFERENCES

Ali, M. S., Kshirsagar, M., Naredo, E., and Ryan, C. (2021). AutoGE: A Tool for Estimation of Grammatical Evolution Models. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, pages 1274–1281. SCITEPRESS - Science and Technology Publications.

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18.

Fagan, D. and Murphy, E. (2018). Mapping in Grammatical Evolution. In Ryan C., O'Neill M., C. J., editor, *Handbook of Grammatical Evolution*, pages 79–108. Springer International Publishing, Cham.

Gang, W. and Soule, T. (2004). How to Choose Appropriate Function Sets for Genetic Programming. In Keijzer, M., O'Reilly, U.-M., Lucas, S. M., Costa, E., and Soule, T., editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 198–207, Coimbra, Portugal. Springer-Verlag.

Harper, R. and Blair, A. (2006). Dynamically Defined Functions In Grammatical Evolution. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 9188–9195, Vancouver. IEEE Press.

Koza, J. R. (1993). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

Kshirsagar, M., Jachak, R., Chaudhari, P., and Ryan, C. (2020). GEMO: Grammatical Evolution Memory Optimization System. In *Proceedings of the 12th International Joint Conference on Computational Intelligence*, pages 184–191. SCITEPRESS - Science and Technology Publications.

Megane, J., Lourenco, N., and Machado, P. (2021). Probabilistic Grammatical Evolution. In Hu, T., Lourenco, N., and Medvet, E., editors, *EuroGP 2021: Proceedings of the 24th European Conference on Genetic Programming*, volume 12691 of *LNCS*, pages 198–213, Virtual Event. Springer Verlag.

Murphy, A. and Ryan, C. (2020). Improving module identification and use in grammatical evolution. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7.

Nicolau, M. and Agapitos, A. (2018). Understanding Grammatical Evolution: Grammar Design. In Ryan, C., O'Neill, M., and Collins, J. J., editors, *Handbook of Grammatical Evolution*, pages 23–53. Springer International Publishing, Cham.

Nicolau, M. and Agapitos, A. (2021). Choosing function sets with better generalisation performance for sym-

bolic regression models. *Genetic Programming and Evolvable Machines*, 22(1):73–100.

Nicolau, M., Agapitos, A., O'Neill, M., and Brabazon, A. (2015). Guidelines for defining benchmark problems in Genetic Programming. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1152–1159. IEEE.

Oliveira, L. O. V. B., Martins, J. F. B. S., Miranda, L. F., and Pappa, G. L. (2018). Analysing symbolic regression benchmarks under a meta-learning approach. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1342–1349, New York, NY, USA. ACM.

O'Neill, M. and Ryan, C. (2000). Grammar based function definition in Grammatical Evolution. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 485–490, Las Vegas, Nevada, USA. Morgan Kaufmann.

Ortega, A., de la Cruz, M., and Alfonseca, M. (2007). Christiansen grammar evolution: Grammatical evolution with semantics. *IEEE Transactions on Evolutionary Computation*, 11(1):77–90.

Patten, J. V. and Ryan, C. (2015). Attributed Grammatical Evolution using Shared Memory Spaces and Dynamically Typed Semantic Function Specification. In Machado, P., Heywood, M. I., McDermott, J., Castelli, M., Garcia-Sanchez, P., Burelli, P., Risi, S., and Sim, K., editors, *18th European Conference on Genetic Programming*, volume 9025 of *LNCS*, pages 105–112, Copenhagen. Springer.

Raymond, C., Chen, Q., Xue, B., and Zhang, M. (2020). Adaptive weighted splines: a new representation to genetic programming for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 1003–1011, New York, NY, USA. ACM.

Ryan, C., Collins, J., and O'Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In Banzhaf, W., Poli, R., Schoenauer, M., and Fogarty, T., editors, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1391, pages 83–96. Springer, Berlin, Heidelberg.

Ryan, C., O'Neill, M., and Collins, J. (2018). Introduction to 20 Years of Grammatical Evolution. In *Handbook of Grammatical Evolution*, pages 1–21. Springer International Publishing, Cham.

Ryan, C., Rafiq, A., and Naredo, E. (2020). Pyramid: A hierarchical approach to scaling down population size in genetic algorithms. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8.

Uy, N. Q., Doan, T. C., Hoai, N. X., and O'Neill, M. (2013). Guiding Function Set Selection in Genetic Programming based on Fitness Landscape Analysis. In *GECCO 2013 - Companion Publication of the 2013 Genetic and Evolutionary Computation Conference*, number 2, pages 149–150.

Wong, T. and Yeh, P. (2020). Reliable Accuracy Estimates from k-Fold Cross Validation. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1586–1594.