Managing Repetition in Grammar-Based Genetic Programming

Miguel Nicolau
College of Business
University College Dublin, Ireland
miguel.nicolau@ucd.ie

Michael Fenton
College of Business
University College Dublin, Ireland
michael.fenton@ucd.ie

ABSTRACT

Grammar-based Genetic Programming systems are capable of generating identical phenotypic solutions, either by creating repeated genotypic representations, or from distinct genotypes, through their many-to-one mapping process. Furthermore, their initialisation process can generate a high number of duplicate individuals, while traditional variation and replacement operators can permit multiple individuals to percolate through generations unchanged. This can lead to a high number of phenotypically identical individuals within a population. This study investigates the frequency and effect of such duplicate individuals on a suite of benchmark problems. Both Grammatical Evolution and the CFG-GP systems are examined. Experimental evidence suggests that these useless evaluations can be instead be used either to speed-up the evolutionary process, or to delay convergence.

CCS Concepts

•Mathematics of computing → Evolutionary algorithms; •Computing methodologies → Genetic programming;

Keywords

Genetic Programming; Fitness evaluation; Speedup technique; Running time analysis

1. INTRODUCTION

Evolutionary algorithms rely on populations of candidate solutions to probe the search space, but also on a convergence process to focus the search in promising areas. Depending on their rate of convergence, this can result in combination and variation operators generating similar solutions, with an associated probability of generating previously seen solutions.

This is particularly the case for Genetic Programming (GP) [12] systems. As the solutions generated are combinations of symbols that obey a pre-defined syntax, the solution space is discrete in nature (albeit potentially very large or even infinite). Thus, without any mechanisms in place, previously evaluated solutions can reappear. Depending on the problem being solved, the associated cost of evaluating candidate solutions (time and/or other resources) might be a cost worth minimising.

In the Grammatical Evolution system (GE) [19], this effect can be even more pronounced. GE typically explores a linear representation of integers, which are mapped into syntax-respecting solutions through the use of a grammar, by mapping integers to the number of choices associated with grammatical symbols. This means that multiple integers can map to the same grammar choices, thus giving rise to a many-to-one mapping process, and the exploration of neutral landscapes [2].

This is also the case with Context-Free Grammar GP (CFG-GP) [25], to a lesser extent. The CFG-GP approach evolves derivation-tree structures, created using a grammar. Depending on the grammar, several different derivation trees result in the same phenotypic solution, thus employing a many-to-one mapping process as well.

In this study, both GE and CFG-GP are examined with respect to their propensity to generate repeated solutions. A series of symbolic regression experiments were run to test several approaches to manage repetition. The results show that detection of repeated solutions can provide a substantial reduction in fitness evaluations, without an adverse effect in algorithm performance.

This paper is structured as follows. Section 2 gives an overview of the literature in the area. Section 3 presents the repetition-avoidance approaches used in this study. Section 4 provides details of datasets used, global as well as specific setups for both systems, grammar design, and re-initialisation approaches. The results obtained from all experiments are discussed in Section 5, and finally Section 6 draws conclusions and recommendations for future work.

2. RELATED WORK

Tabu Search (TS) is a local search technique devised by Glover [5]. It generates a neighbourhood around solutions consisting of other solutions created using a single step of a variation operator. From a neighbourhood around a generator, the best solution not yet in a fixed-length tabu list t is chosen as the new generator, and the previous generator is appended to t. Measures have to be taken if all neighbours of the current generator are already listed in t.

There have been many studies combining a Genetic Al-

GECCO '16, July 20-24, 2016, Denver, CO, USA
© 2016 ACM. ISBN 978-1-4503-4206-3/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2908812.2908904

gorithm (GA) with TS. Glover et al. [6] devised the Scatter Search approach, combining the global search provided by GAs with the local search capabilities of TS. Miller and Thomson [14] combined a Genetic Algorithm with Tabu Search to restrict evaluations of an expensive fitness function, while also avoiding the over-exploitation of confined search neighbourhoods. Many other studies have combined GAs with TS, mostly as a local search heuristic; Ting et al. [23] give a substantial overview.

TS is harder to combine with GP, although Balicki designed a method called Tabu Programming [1], combining GP and TS by using a short term memory of previously generated solutions, to avoid revisiting the most recently evaluated solutions. This is achieved by modifying the standard GP operators, allowing the creation of neighbourhoods of possible steps around solutions.

Most efforts for dealing with repeated solutions (or part of solutions) in the GP community seem to be addressed as implementation details, and these are detected with the aim to shorten evaluation time, without affecting the evolutionary cycle. Perhaps the largest body of research in this area has been with Subtree caching for GP: Keijzer [11] gives an excellent overview of its implementation, along with other GP implementation techniques for speedup such as vectorised evaluation. Other studies include that of Wong [27], who combined subtree caching with algebraic equivalence hashing, showing significant reduction in CPU time required, without statistically affecting the performance of GP. More recently, Langdon et al. [13] used unlimited tabu lists both at genotypic (syntax tree) and phenotypic (conditional execution of code) level, to substantially reduce compilation and execution time when improving software. Finally, other GP representations such as Cartesian GP have also been analysed in terms of repeated evaluation avoidance [7].

There are a few studies to address the issue of repeated solutions with GE. Ryan and Azad [21] highlighted the propensity of GE towards generating repeated solutions when using random initialisation of the initial population, and proposed a *Sensible Initialisation* approach, based on GP's ramped half-and-half [12]. Harper [8] further studied the distribution of tree depths resulting from both random and ramped half-and-half initialisation in GE, highlighting the high probabilities of repetition. Finally, Hemberg et al. [9] combined GE and an NSGA-II, and used a tabu list containing all previously evaluated individuals, with the aim to both restrict the re-evaluation of repeated solutions, and also to better explore the resulting pareto front.

3. MANAGING REPETITION

Given the propensity for these systems to generate repeated individuals, it makes sense to reduce the (wasted) CPU time required to reevaluate those repeated candidates. This is particularly the case for time-consuming fitness functions, such as regression of very large datasets, compilation of code, or use of complex simulation environments.

In this study, only full syntactically equivalent repeated solutions are examined (e.g. sub-tree equivalence or code simplification is not used); this is in order to keep the approach as general as possible. Note that most of the findings in this study are applicable to tree-based GP as well.

A total of four approaches were examined; each of these was used with both GE and CFG-GP.

3.1 No tabu list (control scenario)

This setup is used as a control scenario. Duplicate individuals are not treated differently, and are monitored purely to observe the total number of occurrences across a run.

3.2 Tabu list fitness lookup

This setup uses a typical tabu or *lookup* list, as often seen in the literature. Every time a solution is generated, the tabu list is checked: if it is not present, it is added, along with its measured fitness; if it is already present, the prerecorded fitness is used instead of full fitness evaluation.

This approach preserves the population dynamics of the original versions of both systems, and thus their convergence rates, while considerably reducing the number of fitness function calls.

3.3 Tabu list as penalty list

This setup also employs a tabu list, but instead of being used to lookup a corresponding fitness value, it is used to penalise repeated solutions. New candidate solutions which already exist in the tabu list are assigned the worst possible fitness (otherwise they are added to the list).

Although repeated solutions are heavily penalised, they remain in the population (albeit with a very poor fitness). Therefore, they still have a small chance of being selected, particularly when smaller tournament sizes are employed. Depending on the evolutionary setup, this can lead to a lower number of fitness evaluations, and/or further exploration of the search space.

3.4 Tabu list as forbidden list

This final setup also employs a tabu list, but it is used to prevent repeated solutions entering the evolutionary cycle. If a solution is found in the list, it is dropped from the population, and the algorithm's initialisation process (detailed in Section 4) is repeatedly used to create a replacement, until a previously unseen solution candidate is generated.

This is the only approach that re-uses the initialisation process during the run, and aims at maximising the diversity of candidate solutions evaluated. It substantially affects the convergence rate of the base systems, and continuously inserts newly generated solutions into the population.

4. EXPERIMENTAL SETUP

In order to test these approaches, both GE and CFG-GP were applied to three recommended symbolic regression benchmarks [26]:

- Keijzer-6 [10]: $y = \sum_i x \frac{1}{i}$;
- Vladislavleva-4 [24]: $y = \frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$;
- Dow Chemical dataset [3].

Both the Keijzer-6 (K6) and Vladislavleva-4 (V4) problems were chosen as they required both interpolation and extrapolation of their training set variable range, and exhibit no erratic response distribution in those ranges [17]: K6 is a relatively easy problem with a single input variable, while V4 is a harder problem, with 5 input variables. The Dow Chemical Dataset (Dow) is a high-dimensionality dataset (57 predictors), and was used for the EvoCompetitions event at the EvoStar 2010 conference. All these problems were attempted using the recommended training and

Table 1: Experimental Setup

GE	CFG-GP	Parameter	Value
x	x	Population Size	500
x	x	Generations	50
x	x	Max. Evaluations	25000
x	x	Initialisation	Sensible
x	x	Initialisation Max. Depth	10
	x	Global Depth Limit	20
x		Initialisation Tails	$0.5 \times l$
x	x	Tournament Size	1%
x	x	Crossover prob.	50%
x	x	Wrapping	OFF
x		Integer mutation prob.	$1.0/l$
	x	CFG-GP mutation	Subtree
x	x	Elitism	1%

test sets, but with a common function set, detailed in Section 4.3 (protected operators were used when required).

Experimental parameters are given in Table 4, with specific system setups detailed in Sections 4.1 (GE) and 4.2 (CFG-GP). As the number of fitness evaluations at the 50th generation is smaller when repeat detection is in place, experiments were allowed to keep running until 25000 fitness evaluations (500×50) were reached. Results are reported both at the end of generation 50, and after 25000 evaluations were performed.

4.1 Grammatical Evolution

GE was setup in a standard manner, with some modifications (seen in many studies in recent literature):

- An integer-string is used as the genotype;
- Integer-mutation is employed;
- Random integer tails are appended to genotype strings at initialisation [18];
- Genetic operators are applied only to the used portions of the genotype[1];
- Non-mapping solutions are assigned worst fitness.

Note that no maximum size control was used for the solutions generated by GE.

4.2 CFG-GP

CFG-GP was setup identically to the GE implementation described above, but with all operations being performed on derivation tree structures rather than linear genomes:

- Standard GP-style subtree mutation is employed [12] whereby a randomly selected subtree is replaced by a new randomly generated subtree from the same root node. Mutated subtrees can have any depth up to a maximum overall tree depth of 20.
- Subtrees are selected at their root non-terminal value. This means that both standard subtree mutation and leaf mutation (whereby a single terminal is replaced with a new randomly selected terminal with the same root non-terminal) are possible.
- All individuals are guaranteed a single subtree mutation event.
- Depth limits are imposed in subtree crossover by only crossing over subtrees such that newly generated individuals will not exceed the depth limit.

[1]The whole linear genotype string may not have been used during the mapping process.

4.3 Grammars

Grammar-based evolutionary algorithms are susceptible to poor performance if a combination of bad initialisation and poorly designed grammars are used [8, 15]. Historically, random initialisation has been a preferred method in many implementations of GE [4, 19, 21], despite its associated downfalls. The example grammar in Figure 1 illustrates this. Since there are only two production choices for the start rule, an average of 50% of all individuals in the first generation will be either `x1` (25%) or `x2` (25%), leading to a very large rate of repeated solutions in the initial population.

```
<e> ::= <o> <e> <e> | <v>
<v> ::= x1 | x2
<o> ::= + | - | * | /
```

Figure 1: Trivial example grammar.

The use of ramped half-half (sometimes called sensible initialisation when applied to GE [21]) improves this situation by generating a variety of trees from a range of given depths [20]. However, depending on the size of the initial population and the ramping range, a large number of duplicate individuals can still be generated at initialisation.

With the use of variation operators, there is again a high chance of duplication occurring. Taking once again the example grammar, a genetic operation that replaces the production `<o> <e> <e>` with `<v>` reduces the resulting phenotype to either `x1` or `x2`, regardless of the size of the original solution, thus increasing the likelihood of generating repeated solutions. This is particularly the case with mutation on GE's linear representation.

Figure 2 shows the grammar used for the K6 experiments; it was designed to minimise biases [15], minimise the number of non-terminal symbols (to reduce the crossover ripple-effect in linear GE [16]), and to be balanced in the sense of choice of producing vs. consuming rules [22, 8]. Grammars for all three problems were designed in this way, using the same operators and functions, but with obvious differences in the number of predictors (`x1..x5` for V4, `x1..x57` for Dow), and with corresponding adapted number of arithmetic operator choices, to re-balance biases.

```
<e>  ::=  + <e> <e> | - <e> <e>
        | * <e> <e> | / <e> <e>
        | sqrt <e> | sin <e> | tanh <e>
        | exp <e> | log <e>
        | x1 | x1
        | <c><c>.<c><c> | <c><c>.<c><c>
<c>  ::=  0 | 1 | 2 | 3 | 4
        | 5 | 6 | 7 | 8 | 9
```

Figure 2: Grammar for K6 experiments.

Table 2: Generalisation (test fitness) results, Linear GE

	Baseline	Lookup		Penalty		ReInit
		Fitness @ Gen 50	Final Fitness	Fitness @ Gen 50	Final Fitness	
K6	0.0024 ±0.0085	0.0024 ±0.0085	0.0050 ±0.0228	0.0026 ±0.0098	0.0029 ±0.0139	0.0162 ±0.1167
V4	0.0480 ±0.0296	0.0480 ±0.0296	0.0696 ±0.1840	0.0431 ±0.0081	0.0435 ±0.0128	0.0452 ±0.0182
Dow	0.1089 ±0.0085	0.1089 ±0.0085	0.1044 ±0.0257	0.1113 ±0.0395	0.1033 ±0.0078	0.1089 ±0.0166

Table 3: Generalisation (test fitness) results, CFG-GP

	Baseline	Lookup		Penalty		ReInit
		Fitness @ Gen 50	Final Fitness	Fitness @ Gen 50	Final Fitness	
K6	0.0065 ± 0.0208	0.0065 ± 0.0208	0.0069 ± 0.0232	0.0060 ± 0.0327	0.0059 ± 0.0356	0.0157 ± 0.0828
V4	0.0433 ± 0.0105	0.0433 ± 0.0105	0.04325 ± 0.0114	0.0441 ± 0.0083	0.0430 ± 0.0079	0.0593 ± 0.0832
Dow	0.1124 ± 0.0056	0.1124 ± 0.0056	0.1103 ± 0.0060	0.1081 ± 0.0073	0.1056 ± 0.0097	0.1115 ± 0.0062

4.4 Re-initialisation of individuals

The repetition management approach described in Section 3.4 does not allow repeated individuals to undergo the effect of genetic operators, and instead replaces them with newly re-initialised solutions. A number of options are available for re-initialising discarded individuals:

1. Re-initialise using full method;
2. Re-initialise using grow method, to the same depth as the solution to replace;
3. Re-initialise using grow method, up to and including a specified maximum depth.

The first of these options seems to be the least desirable. Bloat is a well-document issue within the GP community [20]. Generally speaking, the longer a typical GP system runs, the more likely it is to produce increasingly large individuals. Since the full initialisation method creates individuals where all branches in the derivation tree are at the same depth [20], a discarded individual with one deep node would be replaced with an extremely dense tree. The net effect of this would be to promote bloat in the overall system.

The second option can also be troublesome. Consider again the example grammar in Figure 1. The minimum derivation tree depth of this grammar is 2, with 2 possible outcomes at that depth (`x1` or `x2`). At a depth of 3, `<o> <e> <e>` maps to `<o> <v> <v>`, thus the total number of possible combinations is only $4 \times 2 \times 2 = 16$. Even at depth 4, the total number of distinct possible expressions is only 1280.

In a typically setup evolutionary run (particularly using ramped-half-and-half initialisation), it is reasonable to expect that the majority of solutions will be generated using small derivation trees. Hence, when replacing a repeated phenotype of a small depth, there might be few or no possible solutions of the same depth to replace it with (which have not yet been generated).

The third and final option therefore seems far safer, and consists of re-initialising discarded repeated individuals by randomly generating a tree up to a specified maximum depth. Provided the grammar contains a recursive element, and provided the maximum depth is not trivially low, this should return a non-repeated solution in most instances.

5. RESULTS AND DISCUSSION

All four tabu list approaches (baseline, fitness lookup, fitness penalty and re-initialisation) were used with both GE and CFG-GP, and applied to all three datasets. Tables 2 (GE) and 3 (CFG-GP) show the mean test fitness results of the best final individuals, averaged across 100 independent runs (fitness is minimised). Since the *Lookup* and *Penalty* experiments required a higher number of generations to reach the total number of 25000 fitness evaluations, results are shown at both generation 50 and after 25000 total evaluations.

The results are quite similar between both systems, and indeed similar between the different approaches. This is very interesting, given that the *Lookup* and *Penalty* approaches evaluate far less individuals when reaching generation 50. There is evidence of slight over-fitting in certain experiments, between the results obtained at generation 50 and after 25000 evaluations (note that no overfit preventing approach was used); this makes it hard to draw conclusions on the relative fitness performance of all systems and approaches, and the results obtained are used mainly as a measure of similarity between all approaches.

The number of repeats and runtimes for all systems were also monitored, and are shown in Tables 4 (GE) and 5 (CFG-GP). These results show the large amount of repeated solutions generated by both systems, with some approaches generating close to 50% repeated solutions for GE, and over 65% repeated solutions for CFG-GP (a result of the tendency of the grow initialisation method to create smaller derivation tree structures, as discussed in Section 4.4).

These tables also show the runtime of each experiment (averaged across all 100 runs for each setup); the baseline average runtimes were used as the reference value, for each system. These results show the trade-off of keeping a very large list of previously encountered solutions. The associated memory and lookup time cost, for some setups, is larger than the cost of re-evaluating repeated solutions. Note however that the datasets used are quite small (training set sizes of 50, 1024 and 747 samples for K6, V4 and Dow, respectively), which combined with vectorised evaluation [11], resulted in extremely fast evaluation times[2]. With much larger datasets, or other expensive fitness functions (such as complex simulations or even physical evaluations), one would expect to achieve a much better trade-off.

Also note that the different results obtained with GE and CFG-GP are due to the different representations and operators used. For example, the *Lookup* approach is relatively faster with linear GE than with CFG-GP, when compared to their corresponding baselines; this is because fitness evaluation is relatively slower in GE (where management of linear structures is very fast), whereas it is relatively faster in CFG-GP (where handling derivation-tree structures results in slower combination/variation time). Likewise, the *Reinit* approach is relatively slower with linear GE than CFG-GP; this is because initialisation is a derivation-tree based technique, which is relatively expensive in GE (where all other operations are linear and hence very fast), and relatively very fast in CFG-GP (where all operators are derivation-tree based).

Finally, more effective caching and lookup algorithms can be used [11], to speedup the list-management process.

Figure 3 shows the average number of repeated solutions present in each generation, and the average solution size, for the V4 experiment (results for the K6 and Dow experiments were similar). Size is meant as the number of used codons in GE, and the number of derivation tree nodes associated with production choices for CFG-GP (which are identical measures). Results for the *Lookup* setup are not shown, as they are identical to *Baseline* up to generation 50.

This figure again shows the effect that the two different approaches (linear genotypes for GE, derivation-trees for CFG-GP) can have. The number of repeats in GE shows a sharp increase in the first few generations, a result of both the large number of repeats from its initialisation process, and also the ripple effect of crossover, when applied to widely different solutions (successful mapping events are more likely to generate smaller individuals). As exploitation of solutions begins, with an associated growth in solution size, the number of generated repeated solutions slowly declines.

[2]100 runs of baseline linear GE for the K6 experiments took 101.24s, on a single core of a 4GHz Intel Core i7 processor.

Note as well the effect of repeat management in solution size. As smaller solutions are very common in linear GE, and are more likely to be repeats, they are more likely to be removed from the population with the *Penalty* and *Reinit* approaches, leading to a corresponding increase in average solution size, particularly towards the later generations. This also partly explains the unexpectedly high relative runtime of these approaches with GE, as seen in Table 4 (longer solutions are slower to evaluate).

The results obtained with CFG-GP show substantially different repeat/size dynamics. The number of repeat individuals rises sharply across the first few generations, while the size of individuals (measured as the number of terminal nodes in the tree) continues to rise throughout the evolutionary run (with the exception of the *Penalty* approach for V4 where size variation is small over time). This indicates a very low number of repeats in the initial population, increasing to a near-stable level as search progresses. Note however that the range of both repeated solution numbers and size of solutions is similar for both systems.

6. CONCLUSIONS & FUTURE WORK

Re-evaluation of repeated solutions is a costly business, particularly with slow and/or expensive fitness evaluations; these include large datasets, but also complex computer simulations, expensive real-world experimentation, and even interactive fitness evaluation. Two of the most common grammar-based GP systems, GE and CFG-GP, are particularly prone to produce repeated solutions due to a complex combination of grammar-design, representation, genetic operators, and mapping-process.

This paper introduced three approaches to eliminate the re-evaluation of repeated solutions, for both GE and CFG-GP, all based on an endless tabu list of previously evaluated solutions. The results obtained show the use of the *Lookup* and *Penalty* approaches produce comparable results to their baseline systems, while executing far less fitness evaluations.

There are plenty of possible future work avenues. The approaches presented make use of an infinite tabu list; it would be interesting to use a finite list, with smarter methods of both solution lookup, and of removing solutions from the list, once they have not been encountered for a long time; this would reduce the computational overhead and memory requirements of repeat detection.

The *Reinit* in particular could be vastly improved. Rather than re-creating random solutions, it could sample solutions around the repeated individual, either by small mutations or indeed with a deterministic process (guided by the tabu list), in order to create a similar solution yet unseen.

The *Penalty* approach could also be improved. Assigning a worst fitness value to repeated solutions is an extreme approach, and in the case of linear GE, effectively gives these solutions the same fitness as non-mapping solutions. A better approach might be gradually increasing penalties for repeated solutions.

The list used in this study is only concerned with phenotype solutions. Yet there are many other levels where such a list could potentially be applied: genotypic level for linear GE, and indeed semantic level for both approaches.

Finally, a comprehensive study of population size and dataset size vs. effectiveness of repeat detection should be

Table 4: Repeats and full system runtimes, linear GE

	Experiment	Total individuals	Total repeats	% repeats	Avg. runtime @ Gen 50	Avg. runtime Final
K6	Baseline	2,500,000	1,122,630	44.91%	100%	100%
	Lookup	4,367,000	1,887,727	43.23%	74.98%	215.49%
	Penalty	4,048,500	1,562,002	38.58%	76.53%	195.04%
	Reinit	2,500,000	944,451	37.78%	156.62%	156.62%
V4	Baseline	2,500,000	1,150,794	46.03%	100%	100%
	Lookup	4,376,000	1,903,558	43.50%	65.97%	208.78%
	Penalty	4,067,500	1,585,459	38.38%	79.91%	206.73%
	Reinit	2,500,000	980,736	39.23%	150.67%	150.67%
Dow	Baseline	2,500,000	1,172,004	46.88%	100%	100%
	Lookup	4,632,500	2,145,514	46.31%	74.19%	192.09%
	Penalty	4,310,500	1,844,761	42.80%	74.03%	145.60%
	Reinit	2,500,000	958,797	38.35%	340.72%	340.72%

Table 5: Repeats and full system runtimes, CFG-GP

	Experiment	Total individuals	Total repeats	% repeats	Avg. runtime @ Gen 50	Avg. runtime Final
K6	Baseline	2,500,000	1,035,230	41.41%	100%	100%
	Lookup	4,433,500	1,951,446	43.27%	102.32%	192.29%
	Penalty	3,884,000	1,368,036	35.22%	66.51%	106.13%
	Reinit	2,500,000	1,253,537	50.14%	82.73%	82.73%
V4	Baseline	2,500,000	1,199,971	48.0%	100%	100%
	Lookup	5,115,500	2,603,631	50.9%	98.9%	223.3%
	Penalty	4,255,000	1,740,174	40.9%	56.6%	98.06%
	Reinit	2,500,000	1,645,270	65.81%	73.71%	73.71%
Dow	Baseline	2,500,000	876,812	35.07%	100%	100%
	Lookup	3,976,500	1,461,153	36.74%	100.81%	163.43%
	Penalty	3,694,500	1,176,643	31.85%	75.37%	112.73%
	Reinit	2,500,000	1,035,056	41.40%	92.95%	92.95%

performed, so as to detect the tipping point where these techniques do indeed improve the overall runtime. Such a study should be performed across a much wider range of problems.

Acknowledgments

This research is based upon works supported by the Science Foundation Ireland under grant 13/IA/1850.

7. REFERENCES

[1] J. Balicki. Tabu programming for multiobjective optimization problems. *IJCSNS International Journal of Computer Science and Network Security*, 7(10):44–51, October 2007.

[2] W. Banzhaf. Genotype-phenotype-mapping and neutral variation – A case study in genetic programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature - PPSN III*, volume 866 of *LNCS*, pages 322–332. Springer-Verlag, 1994.

[3] C. D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekart, A. I. Esparcia-AlcÃązar, C.-K. Goh, J. J. Merelo, F. Neri, M. Preuss, J. Togelius, and G. N. Yannakakis, editors. *Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC*, volume 6024 of *LNCS*. EvoStar, Springer, 2010.

[4] M. Fenton, C. McNally, J. Byrne, E. Hemberg, J. McDermott, and M. O'Neill. Automatic innovative truss design using grammatical evolution. *Automation in Construction*, 39:59–69, 2014.

[5] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[6] F. Glover, J. P. Kelly, and M. Laguna. Genetic algorithms and tabu search: Hybrids for optimization. *Computers & Operations Research*, 22(1):111–134, 1995.

[7] B. W. Goldman and W. F. Punch. Reducing wasted evaluations in cartesian genetic programming. In

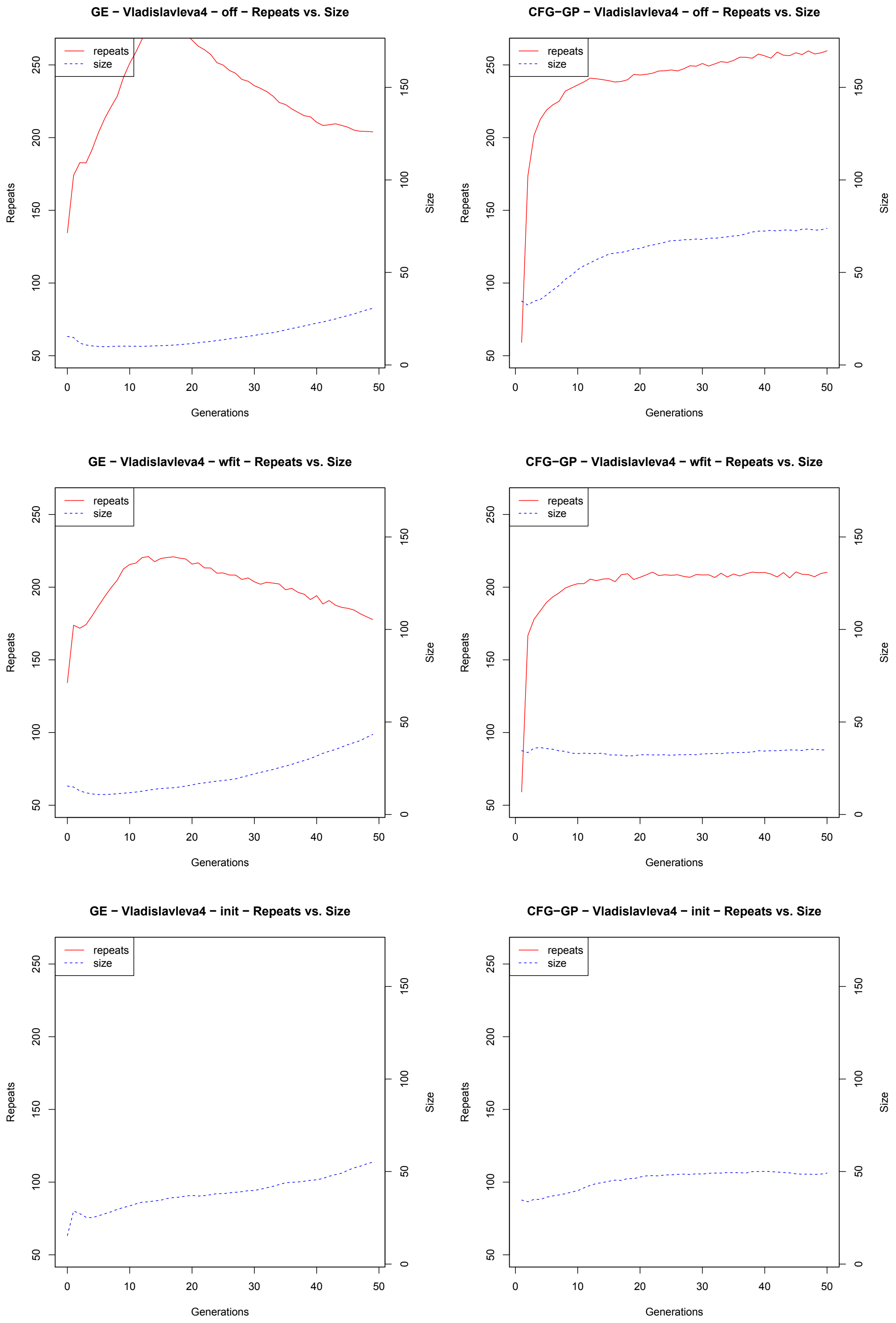

Figure 3: Average number of repeated solutions and solution size per generation, for the V4 experiments. *Baseline* (top), *Penalty* (middle) and *ReInit* (bottom) are shown, for both GE (left) and CFG-GP (right). Results averaged across 100 independent runs.

K. Krawiec, A. Moraglio, T. Hu, S. E.-U. A. and B. Hu, editors, *Genetic Programming, 16th European Conference, EuroGP 2013*, volume 7831 of *LNCS*, pages 61–72. Springer, 2013.

[8] R. Harper. GE, explosive grammars and the lasting legacy of bad initialisation. In *IEEE Congress on Evolutionary Computation, CEC 2010*, pages 2602–2609, 2010.

[9] E. Hemberg, L. Ho, M. O'Neill, and H. Claussen. A symbolic regression approach to manage femtocell coverage using grammatical genetic programming. In N. K. et al., editor, *Genetic and Evolutionary Computation - GECCO 2011*, pages 639–646. ACM, 2011.

[10] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming, 6th European Conference, EuroGP 2003*, volume 2610 of *LNCS*, pages 70–82. Springer, 2003.

[11] M. Keijzer. Alternatives in subtree caching for genetic programming. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, editors, *Genetic Programming, 8th European Conference, EuroGP 2005*, volume 3447 of *LNCS*, pages 328–337. Springer, 2005.

[12] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[13] W. B. Langdon, B. Y. H. Lam, J. Petke, and M. Harman. Improving cuda dna analysis software with genetic programming. In S. Silva, editor, *Genetic and Evolutionary Computation - GECCO 2015*, pages 1063–1070. ACM, 2015.

[14] J. F. Miller and P. Thomson. Restricted evaluation genetic algorithms with tabu search for optimising boolean functions as multi-level and-exor networks. In T. C. Fogarty, editor, *Evolutionary Computing, AISB Workshop, Brighton, UK, April 1-2, 1996, Selected Papers*, volume 1141 of *LNCS*, pages 85–101. Springer, 1996.

[15] E. Murphy, E. Hemberg, M. Nicolau, M. O'Neill, and A. Brabazon. Grammar bias and initialisation in grammar based genetic programming. In A. Moraglio, S. Silva, K. Krawiec, P. Machado, and C. Cotta, editors, *Genetic Programming, 15th European Conference, EuroGP 2012*, volume 7244 of *LNCS*, pages 85–96. Springer, 2012.

[16] M. Nicolau. Automatic grammar complexity reduction in grammatical evolution. In R. P. et al., editor, *Genetic and Evolutionary Computation - GECCO 2004*, 2004.

[17] M. Nicolau, A. Agapitos, M. O'Neill, and A. Brabazon. Guidelines for defining benchmark problems in genetic programming. In *IEEE Congress on Evolutionary Computation, CEC 2015, Sendai, Japan, May 25-28, 2015, Proceedings*, 2015.

[18] M. Nicolau, M. O'Neill, and A. Brabazon. Termination in grammatical evolution: Grammar design, wrapping, and tails. In *IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012, Proceedings*, pages 1–8. IEEE Press, 2012.

[19] M. O'Neill and C. Ryan. *Grammatical Evolution - Evolutionary Automatic Programming in an Arbitrary Language*, volume 4 of *Genetic Programming*. Kluwer Academic, 2003.

[20] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza. *A field guide to genetic programming*. 2008.

[21] C. Ryan and A. Azad. Sensible initialisation in grammatical evolution. In E. C.-P. et al., editor, *Genetic and Evolutionary Computation - GECCO 2003*. AAAI, 2003.

[22] C. Ryan, M. Keijzer, and M. Nicolau. On the avoidance of fruitless wraps in grammatical evolution. In E. C.-P. et al., editor, *Genetic and Evolutionary Computation - GECCO 2003*, volume 2724 of *LNCS*, pages 1752–1763. Springer, 2003.

[23] C.-K. Ting, C.-F. Ko, and C.-H. Huang. Selecting survivors in genetic algorithm using tabu search strategies. *Memetic Computing*, 1:191–203, September 2009.

[24] E. J. Vladislavleva, G. F. Smits, and D. den Hertog. 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, 2009.

[25] P. A. Whigham. Grammatically-based genetic programming. In *Proceedings of the workshop on genetic programming: from theory to real-world applications*, volume 16, pages 33–41, 1995.

[26] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaśkowski, U.-M. O'Reilly, and S. Luke. Better gp benchmarks: community survey results and proposals. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 14(1):3–29, 2013.

[27] P. Wong. Scheme: Caching subtrees in genetic programming. In *IEEE Congress on Evolutionary Computation, CEC 2008, Hong-Kong, June 1-6, 2008, Proceedings*, pages 2678–2685, 2008.