SGE: A Structured Representation for Grammatical Evolution

Nuno Lourenço[1], Francisco B. Pereira[1,2], and Ernesto Costa[1]

[1] CISUC, Department of Informatics Engineering, University of Coimbra, Polo II - Pinhal de Marrocos, 3030 Coimbra, Portugal
[2] Polytechnic Institute of Coimbra, Quinta da Nora, 3030-199 Coimbra, Portugal
{naml,xico,ernesto}@dei.uc.pt

Abstract. This paper introduces Structured Grammatical Evolution, a new genotypic representation for Grammatical Evolution, where each gene is explicitly linked to a non-terminal of the grammar being used. This one-to-one correspondence ensures that the modification of a gene does not affect the derivation options of other non-terminals, thereby increasing locality. The performance of the new representation is accessed on a set of benchmark problems. The results obtained confirm the effectiveness of the proposed approach, as it is able to outperform standard grammatical evolution on all selected optimization problems.

1 Introduction

Evolutionary Algorithms (EA) are computational methods inspired by the principles of natural selection and genetics. Over the years they have been successfully used in different situations, including optimization, design or learning problems. Genetic Programming (GP) is an EA branch that is able to automatically evolve computer programs/algorithmic strategies. One of the most relevant variants of GP is Grammatical Evolution (GE), whose distinctive feature is how it decouples the genotype (a linear string) from the phenotype (a tree expression). GE relies on a mapping process to translate the linear string into an executable program. This transformation is guided by grammar production rules that help to establish the set of syntactically correct programs.

The aim of this paper is to propose Structured Grammatical Evolution (SGE), an enhanced genotypic representation for GE. In SGE there is a one-to-one mapping between genes and non-terminals belonging to the grammar. In order to allow a valid mapping, each gene encodes a list of integers that represent the possible derivation choices of the corresponding non-terminal. The structured representation of SGE, in which a gene is explicitly linked to a non-terminal, ensures that changes in a single genotypic position do not affect the derivation options of other non-terminals. By removing these interactions, SGE might help to solve some well-known locality issues that affect GE [8]. In the next sections we describe the application of SGE to several GP benchmarks

problems [10] and compare its performance against a standard GE approach. The optimization results confirm the effectiveness and efficiency of SGE.

The remainder of the paper is organized as follows: Section 2 provides a brief introduction to GE and reviews relevant contributions dealing with GE representation. Section 3 introduces SGE and details the genotype-phenotype mapping, whereas Section 4 comprises the optimization study. Finally, Section 5 gathers the main conclusions and presents some ideas for future work.

2 Grammatical Evolution

Grammatical Evolution (GE) is a form of Grammar-Based Genetic Programming (GBGP) [5]. As with standard GP, the goal of GE is to evolve executable algorithmic strategies. GE is different from other non grammar-based GP variants, for there is a separation of the genotype, a linear string, and the phenotype, a program in the form of a tree expression. As a consequence, a mapping process is required to map the string into an executable program, using the productions rules of a context-free grammar (CFG). A CFG is a tuple $G = (N, T, S, P)$, where N is a non-empty set of non-terminal symbols, T is a non-empty set of terminal symbols, S is an element of N called axiom, and P is a set of production rules of the form $A ::= \alpha$, with $A \in N$ and $\alpha \in (N \cup T)^*$. N and T are disjoint. Each grammar G defines a language $L(G)$ composed by all sequences of terminal symbols (the words) that can be derived from the axiom: $L(G) = \{w : S \overset{*}{\Rightarrow} w, w \in T^*\}$.

The translation of the genotype into the phenotype is done by simulating a leftmost derivation from the axiom of the grammar. This process scans the linear sequence from left to right and each integer (*i.e.*, each codon) is used to determine the grammar rule that expands the leftmost non-terminal symbol of the current partial derivation tree. Suppose that we have the following production rule,

$$
\begin{aligned}
&< expr > ::= < expr >< op >< expr > && (0)\\
&\quad\quad | (< expr >) && (1)\\
&\quad\quad | < pre - op > (< expr >) && (2)\\
&\quad\quad | < var > && (3)
\end{aligned}
$$

where there are four options to rewrite the left-hand side symbol $< expr >$. In the beginning we have a sentential form equal to the axiom $< expr >$. To rewrite the axiom one must choose which alternative will be used by taking the first codon and dividing it by the number of options for $< expr >$. The remainder of that operation will indicate the option to be used. In the example above, assuming that the first integer is 8, it follows that $8\%4 = 0$ and the axiom is rewritten in $< expr >< op >< expr >$. Then the second integer is read, and the same method is used to the left most non-terminal of the derivation. Sometimes the length of the string is not sufficient to complete the mapping. In those cases the sequence is repeatedly reused in a process known as wrapping. If mapping exceeds a pre-determined number of wrappings, the process stops and the worst possible fitness value is assigned to the individual.

2.1 Other GE Representations

There are some reports in the literature describing enhancements to the standard GE representation and mapping. The *bucket rule* from Keijzer et al. [3] allows a given codon value to select different production choices, thereby removing the bias created by the order of the grammar entries.

In [6], O'Neill et al. presented the Position Independent GE (πGE), an alternative genotype-phenotype mapping. In the traditional GE mapping there is a positional dependency, as the derivation is always performed by expanding the leftmost terminal in the derivation tree. πGE removes this dependency by creating codons with two values: *nont* and *rule*. In this case, *nont* helps to select the next non-terminal NT to be expanded: $NT = nont\%count$, where *nont* is the value present in the genotype, and *count* is the number of non-terminals still in the derivation tree. The *rule* value of the codon pair, as in standard GE, selects which production rule should be applied from the selected non-terminal NT.

Chorus [9] is an alternative proposal aiming at developing a position independent GE, although the results presented in the above mentioned reference do not show any relevant advantage over standard GE.

Fagan and coworkers [1] compared the performance of several mapping mechanisms. Besides the aforementioned πGE and the traditional depth-first expansion they considered two additional methods, breadth-first and a random expansion mechanism, and concluded that πGE provides advantages over standard GE. This result confirms that it is worthwhile to investigate new, alternative, genotypic representations, together with the mapping process.

3 Structured Grammatical Evolution

In SGE each gene is linked to a specific non-terminal and is composed by a list of integers. The length of each list is determined by computing the maximum possible number of expansions of the corresponding non-terminal (see details in section 3.1). This structure ensures that when a gene is modified, it does not affect the derivation options of other non-terminals, thus narrowing the number of changes that occur at the phenotypic level.

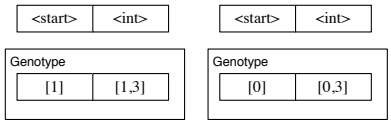
The values that are inside the lists correspond to the number of possible expansion choices. Therefore, when performing the mapping it is possible to remove the modulo rule, thus reducing the redundancy associated with it. Consider the following set of production rules:

$< start >::= < int > \mid < int > * < int >$

$< int >::=1|2|3|4$

There are two non-terminals $\{< start >, < int >\}$. The genotype is composed by two genes, where the first gene is linked to $< start >$, and the second to $< int >$. Then it is necessary to compute the length of the gene's lists by calculating the maximum number of expansions of a non-terminal. The $< start >$ symbol is expanded only once, as it is the grammar axiom. The $< int >$ symbol is expanded, at most, twice, because of the rule $< int > * < int >$. Thus the

lists will have length 1 and 2, respectively. Finally, to fill them we count the number of possible derivation options, c_N, of each non-terminal and assign to each position of the list a random value from the interval $[0, c_N - 1]$. Considering the example above, the $< start >$ symbol has $c_N = 2$ and $< int >$ has $c_N = 4$. Two possible genotypes are depicted in Fig. 1.



(a) Genotype 1 (b) Genotype 2

Fig. 1: SGE: Example of two possible genotypes

The process of translating a genotype into a phenotype is similar to the standard GE mapping. This process starts by expanding the axiom of the grammar, and then expanding the non-terminals in a left-first manner. Consider the example above, where the axiom is the non-terminal $< start >$. To expand it, we look into its gene within the genotype (Fig. 1a). The first unused integer of the list is 1, which selects the option $< int > * < int >$. The next symbol to be rewritten is $< int >$. Its first unused integer is 1, thus it is replaced by the option "2". Next the second $< int >$ is expanded. The first unused integer in the associated gene is 3, which dictates the option "4" should be selected. As there are no more symbols to expand, the process ends, and returns the phenotype: "2*4". The phenotype associated with the genotype of Fig. 1b is "1".

3.1 Pre-Processing

The first step to construct the genotype is to compute an upper bound for the number of times that a non-terminal can be expanded as it defines the list size for each gene. Initially, we iterate through the productions belonging to the grammar, and record the maximum number of references to non-terminals that occur in each choice (Alg. 1). At the same time we build a set that dictates a relation between non-terminals.

Finally, we iterate the set of non-terminals and determine recursively the number of times that, at most, each non-terminal will be expanded (Alg. 2).

Consider the following set of production rules, with $< start >$ as the axiom:

$< start >::= < line > | < line > / < line >$

$< line >::= < var > * < var >$

$< var >::= x1|x2|1$

Using the algorithm described above to compute the size of each gene, we obtain: $< start >: 1, < line >: 2, < var >: 4$. Then we determine the values of c_N, i.e.,

Algorithm 1 Computation of the references that exist in the grammar.

```
countReferences ← {}
isReferencedBy ← {}
for nt in nonTerminalsSet do
    for production in grammar[nt] do
        for option in production do
            if option ∈ nonTerminalsSet then
                isReferencedBy[option] ← nt
                count[option] ← count[option] + 1
            end if
        end for
    end for
    for key in count do
        countReferences[key][nt] ← max(countReferences[key][nt], count[key])
    end for
end for
```

Algorithm 2 Calculate the upper bound for the number of times that a non-terminal can be expanded.

```
function findReferences(nt, isRefBy, countRefProd)
    r ← getTotalReferencesOfProd(countRefProd, nt)
    results ← []
    if nt = startSymbol then
        return 1
    end if
    for ref in isRefBy[nt] do
        result.add(findReferences(ref,isRefBy,countRefProd))
    end for
    references ← references * max(result)
return references
end function
```

the number of derivation choices, for each non-terminal: $< start >: 2, < line >: 1, < var >: 3$.

3.2 Recursive Grammars

The pre-processing described in the previous section does not consider recursive grammars. Standard GE deals with recursion by always trying to perform the translation into an executable program. If it runs out of integers, GE assigns the worst possible fitness value to the individual.

SGE deals with recursion in a different way, as it follows a preemptive approach: a maximum level of recursion must be defined beforehand. Hence it is necessary to introduce a set of intermediate symbols that mimic the levels of the recursion tree. The following example is an excerpt of a grammar for symbolic regression problems:

$$< start >::= < expr >$$
$$< expr >::= < expr >< op >< expr > \mid < var >$$
$$< op >::= + \mid - \mid * \mid /$$
$$< var >::= x$$

Looking into the grammar, we see that the $< expr >$ production is recursive.

Therefore it needs to be rewritten. Assuming that 2 levels of recursion were defined it becomes:

$$
\begin{aligned}
&< start >::= < expr > \\
&< expr >::= < expr\_lvl\_0 >< op >< expr\_lvl\_0 > \\
&\qquad\qquad | < var > \\
&< expr\_lvl\_0 >::= < expr\_lvl\_1 >< op >< expr\_lvl\_1 > \\
&\qquad\qquad | < var > \\
&< expr\_lvl\_1 >::= < var >< op >< var > | < var > \\
&\qquad < op >::= + | - | * | / \\
&\qquad < var >::= x
\end{aligned}
$$

While transforming the grammar we ensure two things: first, that all the symbols have the same probability of being selected after the transformation, because they are copied to each new added level; second, that there will be no invalid individuals, since the mapping process always ends.

All GP variants impose a constraint in the maximum program size, a mandatory step to prevent solutions from growing excessively and becoming computationally intractable. The constraint might be imposed in terms of tree depth, number of available nodes [4], or by imposing limits on the number of wrappings as performed in GE [5]. Following a similar line of procedure, SGE limits the maximum program size by imposing a limit on the number of recursive calls.

3.3 Genetic Operators

GE relies on standard operators to navigate the search space looking for promising solutions to the problem at hand. Two existing variation operators are adapted to work with SGE.

Recombination This operator is an adaptation of the uniform crossover for binary representations. It starts by creating a binary mask with the same length of the genotype. Then the offspring are created by selecting the parents genes based on the mask values. Recombination does not modify the values of the lists inside the genes. Fig. 2 illustrates an application of this operator.

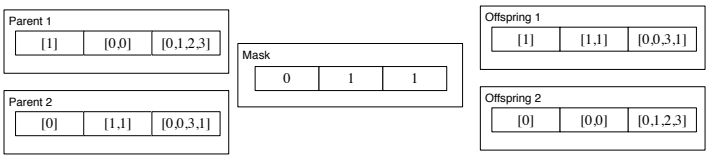


Fig. 2: Application of the recombination operator

Mutation This operator is based on the integer flip mutation. A gene is mutated by randomly selecting a position inside the list and changing it to a new random value from $[0, c_N - 1]$.

4 Experimental Analysis

To validate SGE, three problems were chosen following the guidelines proposed by White et al. to select good GP benchmarks [10]: harmonic curve regression, polynomial regression, and the Santa Fe Ant trail.

4.1 Problems Description

Harmonic Curve Regression The goal is to approximate the series defined by

$$\sum_{i}^{x} \frac{1}{i} \tag{1}$$

where $x \in [1, 50]$. This problem is interesting as it complements the standard interpolation task with a generalisation step. In this second stage, the interval $x \in [51, 120]$ is considered. The production set for the harmonic curve regression is defined as:

$$\begin{aligned}
&< start >::= < expr > \\
&< expr >::= < expr >< op >< expr > |(< expr >) \\
&\qquad\qquad | < pre_op > (< expr >)| < var > \\
&\quad < op >::= + |* \\
&< pre_op >::= + | - |inverse|sqrt \\
&\quad < var >::= x
\end{aligned}$$

where $inverse$ is $1/x$.

Pagie Polynomial This is a hard symbolic regression problem [10], where the goal is to approximate the polynomial function defined by:

$$\frac{1}{1 + x^{-4}} + \frac{1}{1 + y^{-4}} \tag{2}$$

The function is sampled over the range $[-5, 5]$, with a step $s = 0.4$. The production set for this problem is defined as:

$$
\begin{aligned}
&< start >::= < expr > \\
&< expr >::= < expr >< op >< expr > \\
&\qquad\qquad |(< expr >) \\
&\qquad\qquad | < pre\_op > (< expr >) \\
&\qquad\qquad | < var > \\
&< op >::= + \mid - \mid * \mid / \\
&< pre\_op >::= sin|cos|exp|log \\
&< var >::= x|y
\end{aligned}
$$

Artificial Ant The goal is to evolve a strategy that an agent will follow to collect food along the Santa Fe Ant trail. The production set used is the same as in [5].

4.2 Parameters

The GEVA implementation of GE was selected as the baseline of comparison for our experiments. It is an open-source implementation of Grammatical Evolution, in JAVA, and is developed and maintained by O'Neill et al. [7]. SGE was built over the GEVA search engine. There are, however, some slight changes, such as the set of variation operators used and the definition of a maximum level of recursion. The parameters for both SGE and GEVA are defined in Table 1.

We performed 30 independent runs of each approach in the optimization scenarios selected. When comparing SGE with GE a statistical analysis was done to assess if there were differences in the means and, if that was the case, how relevant they were. Since the samples do not follow a normal distribution, the analysis was performed using non-parametric tests. Moreover, and since we are dealing with two unrelated groups, the Mann-Whitney test, at a $\alpha = 0.05$ level of significance, was selected. When differences exist we compute the effect size r[2], to determine how large the differences are. For clarity, we used the following notation: a +++ sign indicates that the effect size is large ($r >= 0.5$), a ++ sign indicates that the effect size is medium ($0.3 <= r < 0.5$), whereas a + identifies a small effect size ($0.1 <= r < 0.3$).

4.3 Results

For the Harmonic Curve Regression, Fig. 3 shows the evolution of the Mean Best Fitness (MBF). An inspection of results shows that the individuals in the initial population of GE have a slightly better fitness, due to the sensible initialization method. The figure also reveals that both GE variants gradually discover better

Table 1: Settings for the Experimental Analysis

Parameter	GEVA	SGE
Initial Population	500	
Recombination rate	0.9	
Mutation rate	0.02	
Replacement	Steady-State with a generation gap of 0.9	
Selection	Tournament with size 3	
Generations	50	
Recombination Operator	Single Point Crossover	SGE Uniform Crossover
Mutation Operator	Integer Flip Mutation	SGE Integer Flip Mutation
Genotype Size	128 (Ramped Half and Half Initialization)	-
Wraps	3	-
Maximum Level of Recursion	-	6

approximations as the run progresses. However SGE exhibits an increased effectiveness, rapidly discovering solutions that surpass the ones found by GE. After 12 generations SGE has already found solutions better than the overall bests of GE.

To estimate the generalization ability, we selected, for each variant (GE and SGE), the best strategy from the initial, middle (gen. 25) and final generations. We then applied the 6 selected strategies to the extended interval from the harmonic curve regression problem. The obtained errors are displayed in Fig. 4. The bars reveal that strategies discovered in later GE and SGE generations tend to obtain better results, suggesting that overfitting did not occur in the interpolation stage. Also, pairs of strategies taken in the same generation (from GE and SGE) obtain comparable results. There are never statistical significant differences, suggesting that, in this particular problem, SGE and GE have similar generalization ability. Finally, it is worth noting that the solutions evolved by SGE seem to be more reliable, as they have a global smaller standard deviation (0.24 vs. 0.4).

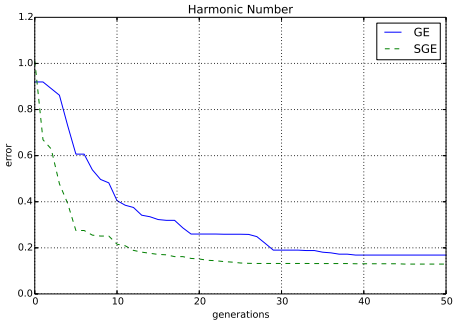


Fig. 3: Mean Best Fitness plots for the Harmonic Number

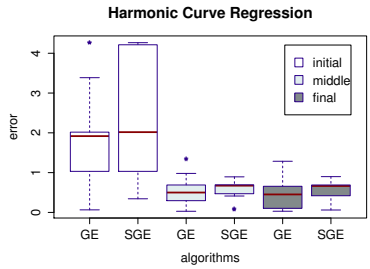


Fig. 4: Mean Best Fitness plots for the Harmonic Curve Regression in the generalization task.

The next problem is the Pagie polynomial. The optimization results follow a trend similar to the one identified in the first problem (Fig. 5). The individuals of the initial population of SGE and GE have comparable fitness. Then, as optimization advances, SGE gradually and consistently obtains low error solutions without stagnating. On the contrary, GE exhibits a slower evolution rate and it stalls at some generations. Looking at the quality obtained by the two variants in the end of the evolutionary run, there is a noticeable difference between SGE and GE. SGE obtained solutions with considerable low error, which reinforces its effectiveness when compared with GE.

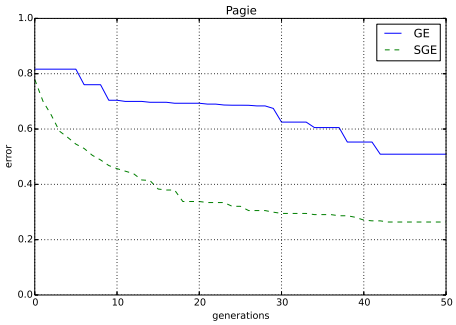


Fig. 5: Mean Best Fitness plots for the Pagie Polynomial

Fig. 6 clearly shows that SGE outperforms GE in the Santa Fe Ant trail, the last selected benchmark. Although the initial solutions of GE have a better quality, at the end of the evolutionary process SGE provides consistently better results. This is so that in all runs, SGE was able to find solutions that allow the ant to eat all the food pieces in the board, leading to a success rate of 100%.

To validate the optimization results, SGE and GE were compared using the statistical tools previously described. The outcomes presented in the column

Statistical Validation of Table 2 reveal that SGE provides statistical significant improvements over the standard GE. We present the p-values obtained, to clarify the magnitude of the differences. The highest p-value is the one for the harmonic experiment, and it still is far from the $\alpha = 0.05$ that was selected as level of significance. We also computed the effect sizes, to assess how large the differences were. The only problem were the effect size is medium ($0.3 <= r < 0.5$) is the harmonic number. In all other problems the effect size is large. These results suggest that SGE is a valid alternative to GE.

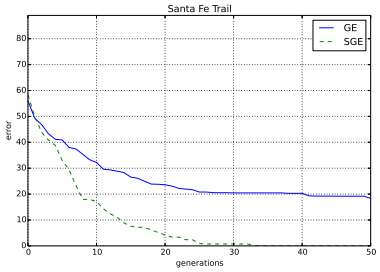


Fig. 6: Mean Best Fitness plots for the Santa Fe Ant Trail

Table 2: Optimization Results: Mean Best Fitness and Standard Deviation over 30 runs

Problem	GE	SGE	Statistical Validation	
			p-value	Effect Size
Harmonic Curve Regression	0.20 (± 0.11)	0.13(± 0.05)	$6.09 * 10^{-3}$	++
Pagie Polynomial	0.50 (± 0.26)	0.29 (± 0.09)	$2.20 * 10^{-6}$	+++
Santa Fe Ant Trail	21.40 (± 12.40)	0.00 (± 0.00)	$9.45 * 10^{-11}$	+++

5 Conclusion

In this paper we proposed Structured Grammatical Evolution (SGE), a new genotypic representation for GE that explicitly considers the features of the grammar being used. The definition of the genotype requires two pre-processing steps: first, recursive productions are rewritten in a non-recursive format, which requires the addition of several new non-terminals; then, an upper bound for the maximum number of non-terminals expansion is computed. After pre-processing is over, the structured genotype is defined. Each gene links to a specific non-terminal and it encodes a list of integers that help to determine the derivation

options during mapping. SGE effectiveness was tested on a set of benchmarks problems and results were encouraging, as it was able to outperform the standard GE representation in all selected problems. Moreover, it proved to be efficient, as it needed a lower number of evaluations to discover good quality solutions

Standard GE has been criticized due to the low locality and extremely high redundancy [8]. One of the goals of the representation proposed in this paper is to enhance GE with a valuable tool to handle these two limitations. We are currently performing a comprehensive set of empirical tests focused on locality. Preliminary results are promising, as they confirm that SGE has higher locality than standard GE [omitted reference]. In the near future we will extend the analysis, in order to gain a deeper insight on how SGE impacts locality and redundancy.

6 Acknowledgments

This work was partially supported by Fundação para a Ciência e Tecnologia (FCT), Portugal, under the grant SFRH/BD/79649/2011.

References

1. Fagan, D., ONeill, M., Galván-López, E., Brabazon, A., McGarraghy, S.: An analysis of genotype-phenotype maps in grammatical evolution. In: Genetic Programming, pp. 62–73. Springer (2010)
2. Field, A.P.: How to Design and Report Experiments. SAGE (2003)
3. Keijzer, M., O'Neill, M., Ryan, C., Cattolico, M.: Grammatical evolution rules: The mod and the bucket rule. In: Foster, J.A., Lutton, E., Miller, J., Ryan, C., Tettamanzi, A. (eds.) Genetic Programming, Lecture Notes in Computer Science, vol. 2278, pp. 123–130. Springer Berlin Heidelberg (2002)
4. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
5. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers, Norwell, MA, USA (2003)
6. O'Neill, M., Brabazon, A., Nicolau, M., Garraghy, S., Keenan, P.: πGrammatical Evolution. In: Genetic and Evolutionary Computation GECCO 2004. vol. 3103, pp. 617–629. Springer Berlin Heidelberg (2004)
7. O'Neill, M., Hemberg, E., Gilligan, C., Bartley, E., McDermott, J., Brabazon, A.: GEVA - Grammatical Evolution in Java. Tech. rep. (2008)
8. Rothlauf, F., Oetzel, M.: On the locality of grammatical evolution. In: Proceedings of the 9th European Conference on Genetic Programming. pp. 320–330. EuroGP'06, Springer-Verlag, Berlin, Heidelberg (2006)
9. Ryan, C., Azad, A., Sheahan, A., O'Neill, M.: No coercion and no prohibition, a position independent encoding scheme for evolutionary algorithms–the chorus system. In: Genetic Programming, pp. 131–141. Springer (2002)
10. White, D.R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jaśkowski, W., O'Reilly, U.M., Luke, S.: Better gp benchmarks: community survey results and proposals. Genetic Programming and Evolvable Machines 14(1), 3–29 (2013)