Benchmarking Individual Representation in Grammar-Guided Genetic Programming

Leon Ingelse, Guilherme Espada and Alcides Fonseca

April 20, 2022

Benchmarking Individual Representation in Grammar-Guided Genetic Programming [⋆]

Leon Ingelse[0000−0001−6067−6318], Guilherme Espada[0000−0001−8128−7397], and Alcides Fonseca[0000−0002−0879−4015]

LASIGE, Faculdade de Ciências da Universidade de Lisboa, Portugal

Abstract. Grammar-Guided Genetic Programming (GGGP) has two popular flavors, Context-Free Grammar GP (CFG-GP) and Grammatical Evolution (GE).

In this paper, we first review the advantages and disadvantages of both GE and CFG-GP established in the literature. Then, we identify three new advantages of CFG-GP over GE: direct evaluation, in-node storage, and deduplication. We conclude with the need to further evaluate the comparative performance of CFG-GP and GE.

Keywords: Grammar-Guided GP · Derivation Trees · Grammatical Evolution.

1 Introduction

Genetic Programming (GP) is praised for its ability to produce useful solutions from vast solution spaces. Being a search-based method, GP performs best when the solution space can be restricted without excluding valid solutions. Grammar-Guided GP (GGGP) [9] uses grammars to solve these issues.

In current research, GGGP has two main approaches for genotype representation. While both approaches use an individual representation that is eventually interpreted as a tree, they differ drastically on the representation. Originally, individuals were represented as derivation trees [9], in a method named Context-Free-Grammar GP (CFG-GP). This approach uses the grammar throughout tree construction, mutation, and/or cross-over operations. The other GGGP approach, called Grammatical Evolution (GE) [7], represents and manipulates individuals as linear strings. Currently, GE is “one of the most widely applied GP methods” [3]. The grammar defined in GE is used to translate these representations to individuals in a process called *genotype-to-phenotype mapping*.

GE has found a strong foothold in GGGP, mainly due to its easier implementation and faster performance of mutation and cross-over operators.

⋆ This work was supported by *Fundação para a Ciência e Tecnologia* (FCT) in the LASIGE Research Unit under the ref. (UIDB/00408/2020 and UIDP/00408/2020) and in the FCT PhD scholarship under ref. (UI/BD/151179/2021), by the CMU–Portugal project CAMELOT, (LISBOA-01-0247-FEDER-045915), and the RAP project under the reference (EXPL/CCI-COM/1306/2021).

In this paper we attempt to compare both approaches. First, we present the advantages of GE over CFG-GP in section 2, such as an easier implementation, faster performance of mutation and cross-over operators and being able to reuse work from the broader field of Evolutionary Computation [3]. Then, we highlight the underexposed advantages of CFG-GP, such as better performance [8], the high locality and the predictable effect of mutation and cross-over operations [6], and the guaranteed validity of created individuals in section 3. Finally, we conclude with a discussion in section 4.

2 Advantages of Grammatical Evolution

CFG-GP is the most direct approach to encoding grammars, as genotype and phenotype are aligned. However, using different representations for the genotype and the phenotype has advantages. Originally, the introduction of GE as a replacement of CFG-GP was mainly motivated by individuals being smaller [7], which reduces memory usage and allows GGGP to be more scalable. Furthermore, there are advantages of GE, due to the *mechanical sympathy* of computers in regard to linear strings. Mainly, GE avoids pointer chasing, making mutation and cross-over operators more efficient. Moreover, the generation of an individual, which effectively consists of generating a random array, can be done faster than the CPU can send the bits to memory.

Correspondingly, on the algorithm implementation side, mutation and cross-over are more easily implemented: Mutation entails the selecting of a random location in the string and updating the value of that location. For cross-over, a location is randomly selected, and two individuals are cut and then concatenated at that location, producing two individuals.

The separation of genotype and phenotype allows the user to "decouple the search engine from the problem [at hand]", so that the same algorithm can be easily applied to different domains [2]. As such, GE also allows the user to reuse research from the areas of Genetic Algorithms and Evolution Strategies [3].

Later, the same authors found populations in GE to be more diverse, because genotypical differences do not necessarily translate to phenotypical differences in GE [4]. As genotypes may contain a lot of redundant information, multiple genotypes can translate to the same phenotype. This phenomenon is called high redundancy. However, this high redundancy is a disadvantage: 90% of mutations do not have any effect on the phenotype, rendering the mutation useless [6].

In the same study, they showed that the mutations that did have effect, often produced child individuals very dissimilar to their parents, resulting in *low locality*. This is because changing the value in one location of a linear string, can cascade to the production of other parts of the individual (fig. 1). High redundancy and low locality result in the fitness of GE resembling a Random Search algorithm [6].

There have been attempts to diminish above-mentioned disadvantages. To make sure genotypical differences affect the phenotype, redundant parts of an individual can be trimmed [7]. By trimming, a maximum length is set for each

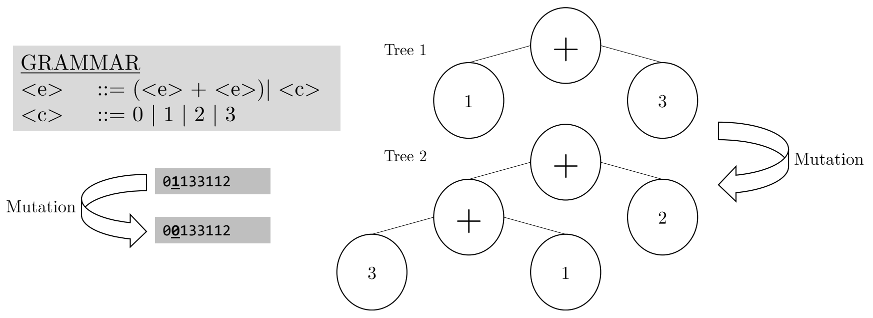

Fig. 1. Example where a mutation cascades to another part of the individual. The second number is mutated, referring to the left node with value 1 in tree 1. This mutation cascades to the right node as well.

linear string. If a genotype, after being operated on, needs more genes than its length allows for, one can apply the *wrapper* approach and continue translation from the start of the linear string [4]. Problematically, it results in a lower locality.

Furthermore, the wrapper approach might result to invalid-individual generation. Look at the grammar fig. 1 and consider the genotype 000. This genotype maps to never-ending plus operations. The fitness of this individual cannot be calculated. Methods such as assigning invalid individuals a low fitness, repairing them, and phenotype-validity checking, are costly and break with the simplicity of GE. Initial populations often show 70% invalid individuals [5].

To improve on the locality of evolved populations, Structured Grammatical Evolution (SGE) was proposed [1]. In SGE, individuals are represented by lists of linear representations. Each list contains all information of one production in the grammar. As such, mutation only affects that single production, and minimizes the cascading to other parts of the individual. A comparison with normal GE shows SGE performing better [1, 2]. Note that SGE breaks with the simplicity of GE, diminishing certain GE advantages, such as a simple implementation. Moreover, individual representations are more complex and take up more memory as every individual requires the space of the largest possible one.

3 Advantages of using Derivation Trees as the Genotype

In section 2, we have presented the three main issues of GE, non-effective mutations, low-locality, and non-valid individuals. CFG-GP does not suffer from these issues as the genotype and the phenotype are aligned. These advantages are put forward to argue that CFG-GP performs better than GE [8, 2].

Besides the direct advantages resulting from the genotype-phenotype alignment, we identify three less discussed advantages of derivation trees: direct evaluation, in-node storage, and deduplication.

Direct Evaluation: During fitness evaluation in GE, each individual is first translated to a string (the program), which is then parsed, and finally evaluated.

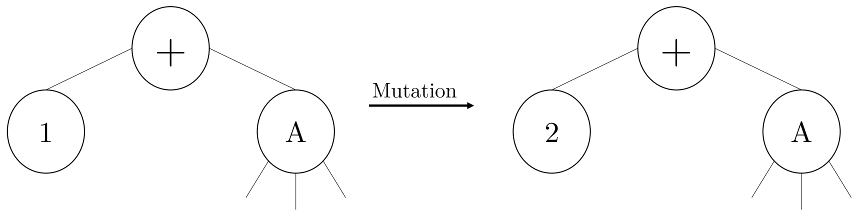

Fig. 2. Example where in-node storage of evaluation is beneficial. The value 1 is mutated to 2. The mutated tree can be easily reevaluated by summing 2 and the evaluation of subtree A.

Crucially, translation and parsing are not free, both in execution time and memory consumption. In languages with a just-in-time compiler, this cost is exacerbated even further, as it has extra work to do tracking freshly generated code. In CFG-GP, derivation trees are traversed only once, evaluating the program directly, with no parsing required, saving time and memory.

In-Node Storage: Since trees consist of nodes in memory, meta-information can be stored in each node. One useful example would be to store the partial evaluation of that node, to reuse in trees that share genomic material with the current one. Caching evaluation results (and other meta-information) is beneficial to a large class of problems, including symbolic regression. Further evaluations can avoid re-evaluating subtrees (c.f. fig. 2).

Deduplication: Because of cross-over, different individuals frequently have subtrees in common, that do not need to occupy duplicated space in memory. Applying deduplication also has the side effect of improving caching (if used in combination with in-node storage) when a given subtree occurs in different individuals. Thus, each subtree needs only be evaluated once, even if it is used in different individuals. This is incredibly difficult in the GE approach: a tree node might not even correspond to an executable element by itself.

4 Discussion

Practitioners should be aware of these advantages when selecting which approach to take. In fact, these decisions are relevant when designing a GGGP framework, and not when implementing a domain-specific grammar. Because one of the main advantages of GE is the ease of implementation of crossover and mutation operators, we argue that a single investment in implementing the CGP-GP operators can be worthwhile as it can be applied to different domains. The second advantage of CGP-GP is the memory saved on the individual representation, but we identify that this might not compensate the extra memory necessary for parsing or the potential time savings by caching evaluation or memory saved by deduplicating common subtrees. Furthermore, despite being less widely used, CFG-GP can perform better than GE [8, 6].

Future research should consider a wide-range comparison of both approaches, covering usability, performance, maintainability, interpretation and scalability.

References

1. Lourenço, N., Assunção, F., Pereira, F.B., Costa, E., Machado, P.: Structured grammatical evolution: a dynamic approach. In: Handbook of Grammatical Evolution, pp. 137–161. Springer (2018)
2. Lourenço, N., Ferrer, J., Pereira, F., Costa, E.: A comparative study of different grammar-based genetic programming approaches. pp. 311–325 (03 2017). https://doi.org/10.1007/978-3-319-55696-3_20
3. McKay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming: a survey. Genet. Program. Evolvable Mach. **11**(3-4), 365–396 (2010). https://doi.org/10.1007/s10710-010-9109-y, https://doi.org/10.1007/s10710-010-9109-y
4. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation **5**(4), 349–358 (2001)
5. ONeill, M., Ryan, C.: Grammatical evolution: Evolutionary automatic programming in a arbitrary language, volume 4 of genetic programming (2003)
6. Rothlauf, F., Oetzel, M.: On the locality of grammatical evolution. In: European conference on genetic programming. pp. 320–330. Springer (2006)
7. Ryan, C., Collins, J., Neill, M.O.: Grammatical evolution: Evolving programs for an arbitrary language. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 1391, pp. 83–96 (1998). https://doi.org/10.1007/BFb0055930
8. Whigham, P.A., Dick, G., Maclaurin, J., Owen, C.A.: Examining the" best of both worlds" of grammatical evolution. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1111–1118 (2015)
9. Whigham, P.A., et al.: Grammatically-based genetic programming. In: Proceedings of the workshop on genetic programming: from theory to real-world applications. vol. 16, pp. 33–41 (1995)