Introduction to 20 Years of Grammatical Evolution

Conor Ryan, Michael O'Neill, and JJ Collins

Abstract Grammatical Evolution (GE) is a Evolutionary Algorithm (EA) that takes inspiration from the biological evolutionary process to search for solutions to problems. This chapter gives a brief introduction to EAs, paying particular attention to those involved in automatic program generation. We then describe grammars, the core building blocks of programs, before detailing how GE's usage of them is one of the key differentiators between it and other EAs.

We give a brief overview of GE and its use, before looking at some of the key developments in the past 20 years, along with a detailed look at the chapters in this book.

1 Evolutionary Computation

Evolutionary Computation (EC) is a machine learning technique inspired by the manner in which the biological evolutionary process operates. Populations of *individuals*, that is, candidate solutions, are evaluated and their performance on a particular problem scored. The population is replaced with a new one created by probabilistically recombining the best performing individuals. In this way, the population slowly evolves towards an optimal or near optimal solution.

Two key factors that limit the sort of problems that can be tackled by EC and, indeed, any iterated machine learning technique, are *representation* and *fitness*. Representation is concerned with the complexity of the solutions that the system can evolve and manipulate. As individuals become more complex, it becomes increasingly more difficult to recombine them with each other.

C. Ryan (✉) · JJ Collins
Department of Computer Science and Information Systems, University of Limerick, Castletroy, Limerick, Ireland
e-mail: conor.ryan@ul.ie; j.j.collins@ul.ie

M. O'Neill
School of Business, University College Dublin, Dublin, Ireland
e-mail: m.oneill@ucd.ie

C. Ryan et al. (eds.), *Handbook of Grammatical Evolution*,
https://doi.org/10.1007/978-3-319-78717-6_1

Fitness is the ability to measure the quality of an individual, specifically its ability to solve the problem at hand. If no *fitness evaluator* exists, creating one can be prohibitively expensive, as unless they are quick and accurate, they will quickly become a bottleneck.

EC has been used with considerable success in areas as varied as Bioinformatics [15, 51], Automatic Circuit Generation [42, 94] and Fluid Dynamics [4]—as far back as the seventies!

Evolutionary Automatic Programming is specifically focused on evolving programs, and more recently has been referred to as the problem of program synthesis. The most commonly used approach, Genetic Programming (GP) [41–45] uses expression trees to represent individuals, as in Fig. 1.

These individuals are recombined with each other using *crossover*, an operation that swaps subtrees from the two parent individuals. The subtrees are selected at random and placed into the corresponding location in the other parent, resulting in two offspring as in Fig. 2.

GP has enjoyed much success and has been successfully applied to an enormous number of problem domains. There is, however, no simple way to deal with multiple types in GP, nor to handle constraints for the manner in which programs are put together. This is because all GP individuals must obey the *closure rule*, that is, all functions must take and return the same type. It is possible to use *Strongly Typed Genetic Programming* [50], in which multiple types can be maintained, but this

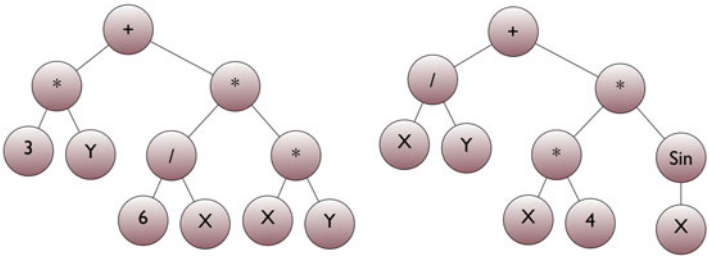

Fig. 1 GP individuals represented as syntax trees. The individual on the left corresponds to **(+ (* 3 Y) (* (/ 6 X) (* X Y)))** while the one on the right corresponds to **(+ (/ X Y) (* (* X 4) (Sin X)))**

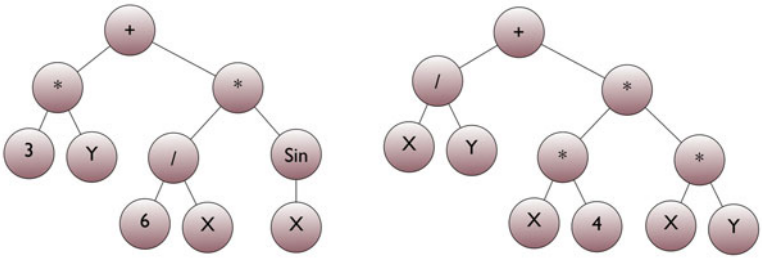

Fig. 2 The resulting offspring from crossing over the parents from Fig. 1

involves performing constrained crossover, with only those nodes of the same type being able to be swapped, which reduces the searching abilities of the system.

However, this constrains the search space, and becomes particularly problematic when dealing with dimensionally aware [37, 38] problems. Furthermore, it also doesn't facilitate the passing of information down through the trees as in Attribute Grammars (AG), which is necessary to generate dynamic types such as those required in matrix multiplication.

Most users do not use GP with multiple types, however, and standard GP has achieved extraordinary success across a very wide range of domains.

2 Grammatical Evolution

2.1 Grammars and Evolutionary Computation

All evolutionary systems that produce programs use grammars of one form or another whether explicitly [49] or implicitly [87]. Grammars describe how programs can be constructed from constituent parts, i.e. how variables and operators can be legally combined to create executable code. The sorts of languages that different kinds of grammars can produce is documented in Chomsky's well known Hierarchy of Grammars [7–9]. Most EC systems use Context Free Grammars (CFG), Type-2 grammars.

As noted above, most Evolutionary Automatic Programming (EAP) systems, including GP, generally considered to be one of the more advanced ones, exclusively use *Closed Grammars* [72, 73], which are a special, restricted form of CFGs that have a single type.

Sometimes these are implicit, as with GP and Gene Expression Programming [23], while other systems are more explicit, such as GE, G3P [95–98], etc. The main trade-off between implicit and explicit grammar usage is speed and expressiveness. We refer the interested reader to two relatively recent syntheses of grammars and genetic programming [49], and more broadly in the context of developmental systems [5].

GE, on the other hand, employs simple linear strings (typically binary or integer) as genotypes, using a mapping scheme to map them onto arbitrarily complex structures. The mapping scheme takes the form of a CFG, which specifies legal relationships between *terminals* (items which can appear in the final structure) and *non-terminals* (interim values to help link terminals together). CFGs enable one to evolve considerably more complex structures than standard GP, because they permit multiple types.

GE has a modular nature, see Fig. 3, meaning that everything from the problem being tackled to the language being used and even the search algorithm being employed can easily be swapped out. Section 4 describes how this modular nature has lead to a massive community effort in further developing GE.

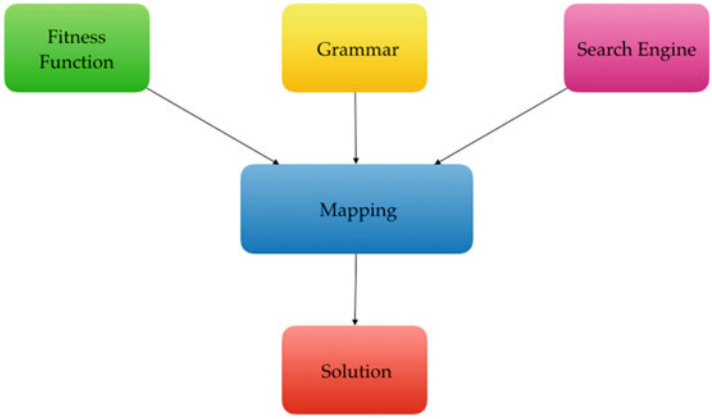

Fig. 3 The modular nature of Grammatical Evolution. Everything from the fitness function to the grammar and even the search engine can be modified or replaced

3 Crash Course in Grammatical Evolution

GE traditionally uses an evolutionary algorithm comprising a variable-length linear genome encoding of a computer program. The genotype-phenotype mapping (mapping) takes as input the linear genome and a grammar, and outputs a sentence in the language described by the grammar, with context-free grammars being the most often used. To drive search the quality of the each individual (that is, a sentence from the language) needs to be assigned a measure of quality.

GE individuals are usually executable entities, but can be any structure represented by a grammar; for example, Chapter 13, "Design, Architecture, and Engineering with Grammatical Evolution" in this book describes the GENR8 [30] system that uses GE and Autodesk's Maya CAD tool to evolve digital surfaces.

When the sentence is in the form of code, it is usually embedded in some wrapper code to manage its execution. The result of the execution of the code is used as its measure of fitness.

We illustrate the mapping process using a simple example grammar to generate strings of characters, vowels and consonants. We first specify the grammar of the output language, which describes all possible sentences that can be generated.

The sentences generated by the example grammar below are of type `string`, which are comprised of one or more `letter`'s. A `letter` is allowed to be one of our primitives, that is, either a `vowel`, `consonant` or `character`.

A convenient formal notation for grammars, often employed by GE, is Backus Naur Form (BNF). BNF is comprised of the tuple {N, T, P, S}, where N is the set of intermediary symbols called non-terminals, which are mapped to the set of terminal symbols (T) according to P, the set of production rules. The terminal set consists of items that can actually appear in legal sentences for the grammar. The final item, S, is a special non-terminal *start* symbol, from which all derivation sequences begin.

For example, in this particular grammar the terminals are neatly described by three types: `vowel`, `consonant` and other `character`. We use the following sets for N and P.

N = {<string>, <letter>, <vowel>, <consonant>, <character>}, and

T = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, ", ?, ',', ., ;, :, ' '}.

That is, the terminal set consists of letters, spaces and punctuation symbols, while the non-terminal set consists of the three types noted above, along with `string`, the start symbol, and `letter`. `letter` is a non-terminal that will be used to help group various vowels, consonants and characters together. The production rules for this grammar can be specified as follows:

```
<string> ::= <letter>|<letter><string>
<letter> ::= <vowel>|<consonant>|<character>

<vowel> ::= a|e|i|o|u
<consonant> ::= b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|y|z
<character> ::= "|*|?|Ă|@|,|.|;|:|' '
```

Thus, the above grammar contains the set of all possible primitive symbols of the sentences, and the structural rules, which govern the generation of syntactically legal sentences. For example, the following is an example of a sentence generated by this grammar, with a partial derivation tree shown overleaf in Fig. 4:

to evolve or not to evolve, that is the question.

3.1 Mapping

GE individuals describe a derivation sequence through a grammar. They do so by selecting choices from the production rules at every derivation step, for example, whether to choose a `vowel`, `consonant` or `letter` from `letter`.

The linear genome is interpreted as 8 bit *codons*, i.e. the smallest functional unit in GE. Each time a choice needs to be made in the derivation sequence a codon is taken and the **mod** of the number of available rules calculated, which is then used to select the appropriate rule. If, for example, we were choosing a production rule for `letter` we would mod the codon by 3 because there are three production rules.

The process continues as described in Fig. 5, consuming a codon for each choice in the derivation sequence, until the full derivation tree has been produced. If there are unconsumed codons remaining, these are said to be the *tail* of an individual and do not contribute to the mapping. In the event that the individual has not fully mapped and all the codons are consumed, either the individual is simply abandoned and assigned the lowest possible fitness or is *wrapped*, meaning that the first codon is reused. In these cases, an upper limit is placed on the number of times an individual can be wrapped.

Although this can lead to more successful mappings, particularly early in runs, results have been mixed [89] and the ability for wrapping to help evolution is often

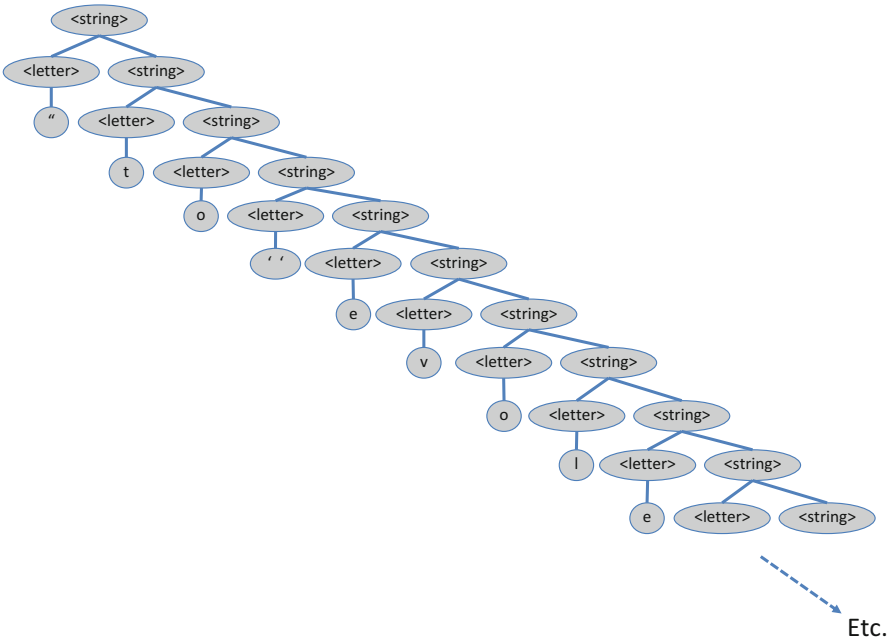

Fig. 4 Partial derivation of the sentence “to evolve or not to evolve, that is the question.” from the example grammar

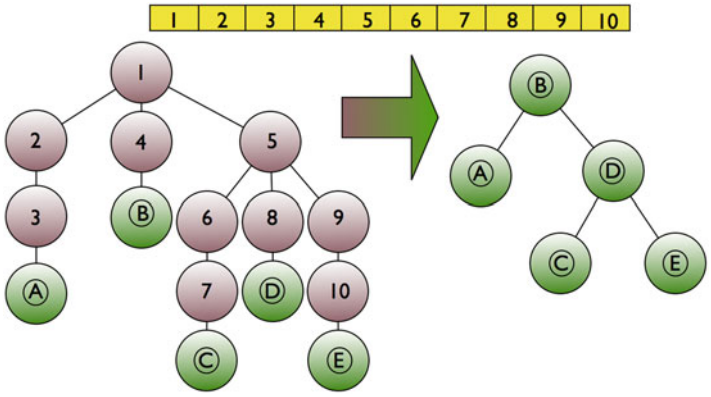

Fig. 5 GE generating a derivation and corresponding parse tree from a binary string. The numbers indicate the order of the mapping was done; circled nodes labelled with letters indicate terminals

dependent on the grammar being used. Many researchers have found that removing wrapping doesn't have a major detrimental effect.

3.2 *Alternative Grammars*

With the exceptions noted here, GE and, indeed, virtually all other grammar based systems, predominantly use CFGs which, although expressive enough for GE to be a very broadly applicable system [3, 24, 61, 71], is limited to regular and context free languages.

Alternative grammars, which have been employed with GE include Attribute Grammars (more details below in Sect. 3.2.1), Shape Grammars [77], L-Systems [66] and Map L-Systems [83], logic grammars [39], graph grammars [47], meta-grammars (albeit CFG) [63, 70] and Tree Adjoining Grammars [54, 56, 58].

3.2.1 Attribute Grammars

Attribute Grammars (AG) can be used to expand the expressive power of GE by attaching *attributes* (pieces of information) to the symbols in a grammar [10, 11, 35, 36, 75, 84]. These entities can interpret and generate attributes; attributes are generated either passed down (*inherited*) or passed up (*synthesized*), although *default* attributes can also be created and passed around as in Fig. 6. Attributes can take any form, from simple atomic forms to arrays or lists. These attributes can be used by a developing structure in AGE to pass information about various parts of the structure to other parts. AG facilitates the manipulation and exploitation of contextual information, which can be about other parts of the solution or the problem. For example, in the context circuit design, attributes could be used to pass information about which input pins have already been processed.

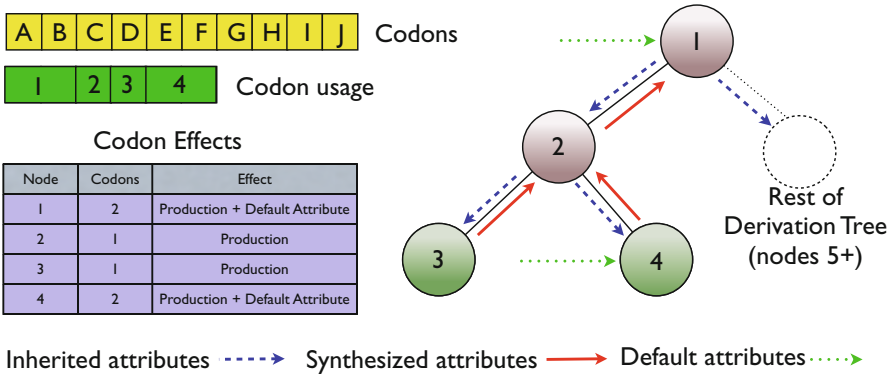

Fig. 6 The GE mapping process augmented with AG. Individuals are mapped from simple binary strings (*codons*) to high level structures using arbitrarily complex grammars, including attribute grammars, which can pass contextual information around

4 Twenty Years of Grammatical Evolution

As shown in Fig. 3, the genotype-phenotype mapping of GE provides the advantage of a modular framework to approach Genetic Programming. The main components of this framework are the search engine, the mapper, the grammar, and the fitness evaluation. Activities over the past 20 years can be described in terms of these components, see Fig. 7 for an overview.

Research in the search engine revolves around understanding the impact of the genome encoding [34], initialisation [59, 86], modularity [29, 33, 82, 90–93], crossover [27, 48, 69, 72], the impact of dynamic environments [13, 79], the behaviour of search operators of crossover and mutation, proposing alternative search operators [6, 17, 25, 28, 52, 62, 76] to replacing the traditional evolutionary algorithm with alternatives such as Particle Swarm Optimisation [65, 74], Simulated Annealing [85], Differential Evolution [64] and even random search [85]. Continuing this vein of research Chapter 7, "Geometric Semantic Grammatical Evolution" outlines a geometric semantic search operator approach to GE, and in Chapter 3, "On the Non-uniform Redundancy of Representations for Grammatical Evolution: The Influence of Grammars" we see an emphasis on analysing the locality of the GE mapping and some of its genetic search operators.

The mapping process itself has been a target for investigation with a number of alternatives having been proposed in part to gain a deeper understanding of the generative process and in attempts to make improvements by, for example, complexifying the mapping by bringing it closer to its biological counterpart [1, 16, 40]. Chapter 4, "Mapping in Grammatical Evolution" provides an overview and highlights key studies in this area.

At the heart of GE is the grammar and while the majority of papers adopt CFGs, we noted earlier in this chapter (Sect. 3.2) the variety of grammars which have been adopted with a GE mapping is impressive, including shape, logic, attribute, meta, graph and tree adjunct grammars to prefix, infix and postfix encoding [31]. This line of research continues to this day, and Chapter 2, "Understanding Grammatical Evolution: Grammar Design" provides a critical analysis on the importance of grammar design in the successful application of GE.

Part of the attraction of genetic programming algorithms such as GE are their flexibility of application. As such, GE has enjoyed application to a wide set of problems areas. Part II of this book contains a selection of chapters highlighting some of these including Financial Modelling (Chapter 11, "Grammatical Evolution in Finance and Economics: A Survey"), Medicine and Bioinformatics (Chapter 15, "Identification of Models for Glucose Blood Values in Diabetics by Grammatical Evolution" and Chapter 16, "Grammatical Evolution Strategies for Bioinformatics and Systems Genomics"), Architecture and Design (Chapter 13, "Design, Architecture, and Engineering with Grammatical Evolution"), Business Analytics (Chapter 19, "Business Analytics and Grammatical Evolution for the Prediction of Patient Recruitment in Multicentre Clinical Trials"), Computational Creativity (Chapter 14, "Grammatical Evolution and Creativity") and Game Artificial

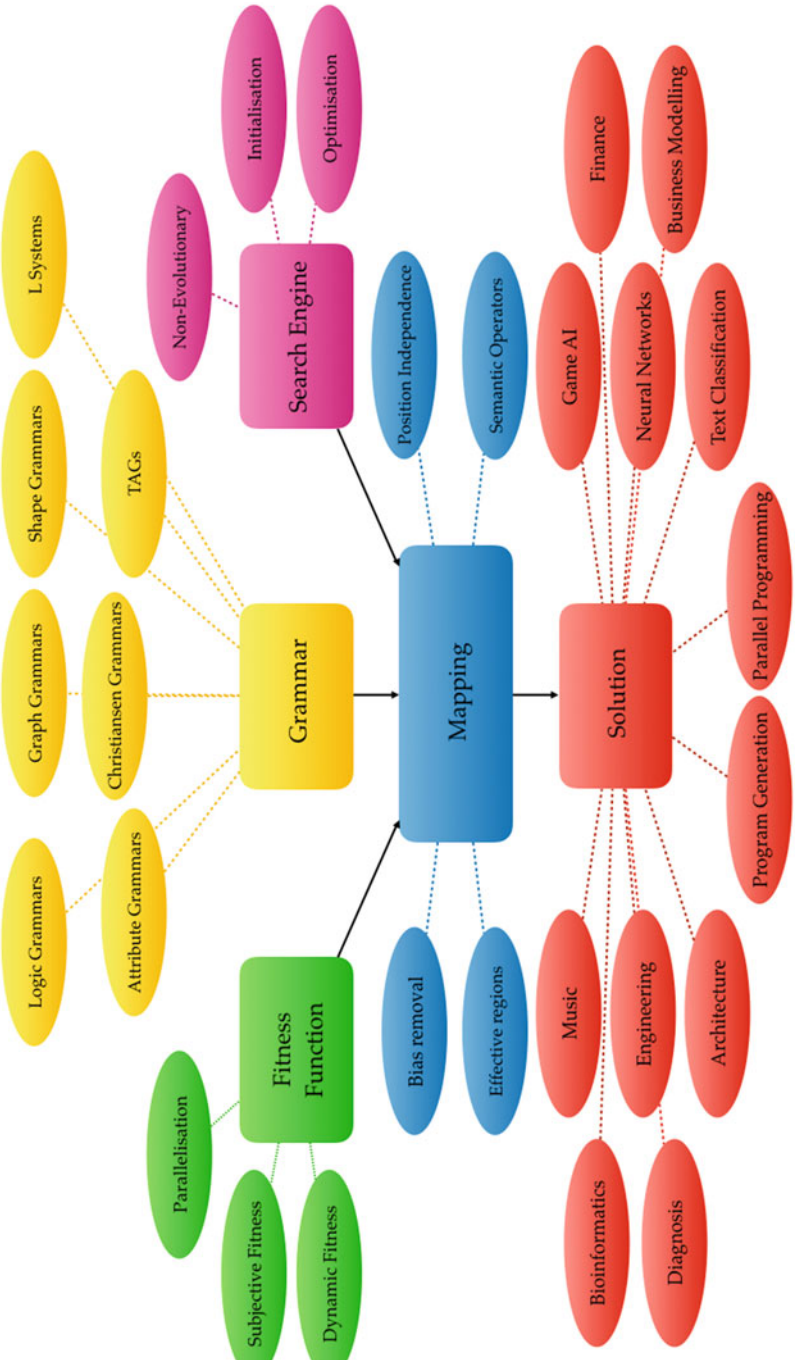

Fig. 7 Some of the key research areas in GE. New topics, especially applications are being added all the time

Intelligence (Chapter 18, "Evolving Behaviour Tree structures using Grammatical Evolution"). Other examples include communication networks [19, 20, 32], search-based software engineering [12] and program synthesis [67, 68, 80], sport analytics [81], eco-system modelling [14, 60], and animation [53, 55, 57].

Matlab[1] and more recently in Python with PonyGE2 [21, 22] with the majority of these employing an integer genome encoding as standard.

Finally, as noted in Sect. 5.4, a large number of variants of GE have appeared. These include position independent approaches such as Chorus and πGE [2, 16, 88], context-sensitive approaches such as Adaptive Logic Programming [39], TAGE [56] and DTAGE [58], to a novel 3D MAP L-system GENr8 [83], and exploiting meta-grammars for Grammatical Evolution by Grammatical Evolution [70].

5 The State of the Art

As noted in the foreword, as the authors of the original GE paper, one of the most rewarding things we have experienced is how it has been taken up by other researchers. The state of the art in 1998 was easy to articulate; there were only three researchers, three problem domains and one system. Twenty years of evolution have had their impact on the sort of applications that GE can be applied to. It is important to note that it isn't possible to definitively state what set up is the best GE, mainly because of the hugely broad spectrum of uses. Instead, we focus in this section on the sorts of choices that need to be considered when tackling a problem with GE, and discuss how various characteristics of problems influence these choices.

5.1 Grammars

AGs are more expressive than CFGs and can be used to enforce constraints and pass context information around derivation trees, as in Fig. 6. The key advantage for CFGs is their simplicity, and mappers using CFGs are generally faster than their AG counterparts, but at the cost of sacrificing expressiveness. Several chapters in this book look in detail at grammar design and our advice is to use the most powerful grammar necessary, but no more.

[1] http://ncra.ucd.ie/Software.html.

5.2 Genetic Operators

The genetic operators are generally inherited from whatever underlying GA or search engine is driving GE, but early work analysing the operation of single point crossover [72] showed that, when compared to other crossover operators, including highly tuned homologous[2] operators, actually performed surprisingly well, giving a very good performance-to-cost ratio. More recent work, such as Chapter 7, "Geometric Semantic Grammatical Evolution" in this book, has examined semantic crossover operators, and show some very promising results.

For a simple GE, set up we recommend what has become known as *effective crossover*, that is, to simply ensure that at least one crossover point is selected within the coding part of an individual as described in Fig. 8. This is simple to implement and dramatically increases the probability that at least one of the offspring will be phenotypically different from the parents.

5.2.1 Initialisation

Originally, we used random initialisation for the GE population. However, as noted in [18, 26, 86], random initialisation can lead to very heavily biased initial populations. Consider the simple grammar below:

$< S >::=< op >< v >< v > \mid < v >$
$< op >::= + \mid - \mid * \mid /$
$< v >::= x \mid y$

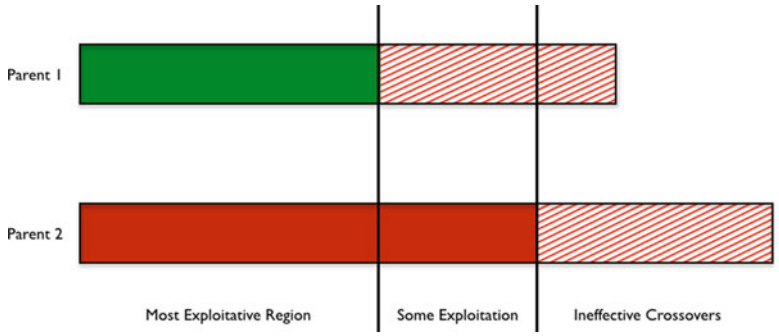

Fig. 8 The three distinct crossover regions for Grammatical Evolution. The solid area in each parent represents the coding regions, while the diagonal lines represent regions that were not used in the mapping. When each crossover point occurs within these regions, the operation will simply result in two offspring identical to the parents. When the points are in either of the other two regions, the crossover operation is said to be *effective*

[2]Crossover operators that attempt to swap functionally similar sections from parents.

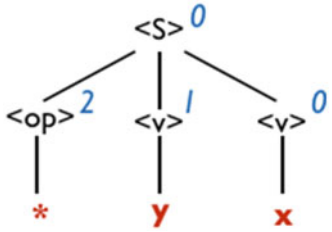

Fig. 9 Creating derivation trees in Sensible Initialisation. The production rule number at each step is noted and will subsequently be used in the following "umod" step. Each individual has a sequence of choices associated with it. In this case the sequence is 0210

Uniform random initialisation will create a population in which 50% of the individuals consist of just one item, due to the $< S >::=< v >$ production; of these, approximately half will be **x** and the rest **y**. Clearly this compromises the variation in the initial population, making evolution towards a useful product more difficult than it needs to be.

Thus, it is important to ensure a spread of individuals in the first generation. Sensible Initialisation [86] takes the *ramped-half-and-half* approach often used in GP and uses it for GE. Sensible initialisation operates by creating a population of derivation trees of various shapes and sizes and performing an "unmod" operation on them to generate linear strings that can subsequently be processed by GE.

When creating each individual in the initial population, first a derivation tree of a particular size is generated. Figure 9 gives an example of a derivation tree of depth 3. The choice made at each step is noted, for example, the initial step used the production rule $< S >::=< op >< v >< v >$, which is choice 0 from those available for $< S >$. Similarly, when mapping $< op >$, choice 2 is made from the available productions rules, that is, $< op >::= *$.

Each individual in the initial population has a list of these choices, which can be used to quickly identify duplicates. Once we are satisfied that the population consists of unique genotypes, the final "unmod" step can be performed.

Unmod produces the actual codons that will be used and essentially performs the opposite operation to mod, returning a number that, when divided by the number of choices available for the particular non-terminal, will return the choice made. In our example, we wish to perform 2 unmod 0 for the first production rule, meaning that we require a number that, when modded by 2 will yield 0.

This means that any even number between 0 and 255 will suffice. Similarly, in the second production rule, we perform 4 unmod 2 as there are four choices and our tree used the second one. Any number in the set {2, 6, 10..} will give the necessary number.

Clearly, unmod is a stochastic operator and, while its output doesn't impact the initial generation in any way, it is crucial to introduce variation so that when individuals from the first generation are crossed over with each other, codons that

end up being used for different production rules than they originally were, will not bias the choices made.

More recent work on initialisation includes that of Nicolau, who demonstrated that across the problems examined in their study, a variant of Harper's PTC2 consistently outperforms other initialisations [59], as well as the work of Lourenco [46], which is further advanced in Chapter 6, "Structured Grammatical Evolution: A Dynamic Approach".

What is crucial though, is to put some effort into ensuring good variation in that initial population, and to avoid simple random initialisation.

5.3 Parameter Settings

As with all EAs, GE has a number of parameter settings, such as population size, mutation rates and the like. There is a vast amount of literature in the field about how to set these parameters, but suffice it to say that population size is the most sensitive, and that more difficult problems generally require larger populations. It is important to turn this knob carefully though, as grammars and initialisation also play a part.

5.4 Variants

As described earlier in Sect. 4, not only has there been considerable research into the use of GE and analysis into its operations, there have also been quite a number of variants. It would take a whole other book to exhaustively test these against each other on a broad enough range of problems to be able to make any sort of recommendations, but readers are encouraged to investigate these variants, particularly those that have been shown to outperform GE on problems related to their own.

6 Contents of This Book

The book is divided into two key sections, *Analysis* and *Applications*. Rather appropriately, we start in the applications section with two chapters on grammar design. In Chapter 2, "Understanding Grammatical Evolution: Grammar Design", Nicolau and Agapitos present some domain-independent guidelines for designing grammars, and, in Chapter 3, "On the Non-uniform Redundancy of Representations for Grammatical Evolution: The Influence of Grammars", Schweim, Thorhauer and Rothlauf present a fascinating study on the impact of grammar design and redundancy on the creation of biased trees.

These are followed up by a trio of chapters on mapping in GE. Starting with a comprehensive survey in Chapter 4, "Mapping in Grammatical Evolution" by Fagan and Murphy, we then move to a contribution by Hemberg, Chapter 5, "Theory of Disruption in GE" in which he formalizes and analyzes the mapping process. This leads nicely into Chapter 6, "Structured Grammatical Evolution: A Dynamic Approach" by Lourenco et al., in which they further develop their *Dynamic Structured Grammatical Evolution* (DSGE) system, a version of GE that employs a different mapping to improve the power of the genetic operators.

Similar motivations are evident in Chapter 7, "Geometric Semantic Grammatical Evolution" and Chapter 8, "GE and Semantics", by Moraglio et al. and Echeandia et al., respectively, the former which develops a semantic crossover operator for GE and the latter employs grammars to enable semantics, giving an excellent review of related work as it does so.

This section of the book is rounded out by two final chapters. Chapter 10, "Comparing Methods to Creating Constants in Grammatical Evolution" by Azad and Ryan tackles the issue of constant generation, highlighting the pros and cons of the more well-known methods, while Dufek et al. describe a parallel implementation of GE in Chapter 9, "Multi- and Many-Threaded Heterogeneous Parallel Grammatical Evolution", which yields hugely impressive results.

We then switch gear to applications and provide seven radically different problems that have been tackled by experts in the field. Starting with a survey of financial applications in GE in Chapter 11, "Grammatical Evolution in Finance and Economics: A Survey" by Brabazon, we move to parallel program generation in Chapter 12, "Synthesis of Parallel Programs on Multi-Cores" by Chennupati et al.

The creative side of GE is explored in the next two chapters, starting with Fenton et al. in Chapter 13, "Design, Architecture, and Engineering with Grammatical Evolution", who use GE to evolve physical designs, and then with Loughran in Chapter 14, "Grammatical Evolution and Creativity" who, in a very philosophical paper, uses GE to evolve music. There then follow two chapters from medical domains; first Hidalgo et al. use GE to generate models for glucose blood values in Diabetics in Chapter 15, "Identification of Models for Glucose Blood Values in Diabetics by Grammatical Evolution", while Moore and Sipper give a thorough review of the use of GE in bioinformatics and systems genomics.

7 Summary

We hope this book provides useful snapshots of research and applications in Grammatical Evolution which has taken place over the past 20 years since the original work was published in EuroGP 2008, and presents some of the state of the art and current thinking in this field. Grammatical Evolution as a form of Genetic Programming in particular in its application to automatic programming or program synthesis has still a lot of open issues to address [78] and we hope to witness and be involved in the continued development of this exciting field of research for some time to come.

References

1. R.M.A. Azad, A position independent evolutionary automatic programming algorithm - the Chorus system, in *Graduate Student Workshop*, New York, 8 July 2002, ed. by S. Luke, C. Ryan, U.-M. O'Reilly (AAAI, Menlo Park, 2002), pp. 260–263
2. R.M.A. Azad, A position independent representation for evolutionary automatic programming algorithms - the Chorus system. Ph.D. Thesis, University of Limerick, Ireland, Dec. 2003
3. R.M.A. Azad, A.R. Ansari, C. Ryan, M. Walsh, T. McGloughlin, An evolutionary approach to wall sheer stress prediction in a grafted artery. Appl. Soft Comput. **4**(2), 139–148 (2004)
4. H. Beyer, H. Schwefel, Evolution strategies - a comprehensive introduction. Nat. Comput. **1**(1), 3–52 (2002)
5. A. Brabazon, M. O'Neill, S. McGarraghy, *Natural Computing Algorithms* (Springer, Berlin, 2015)
6. J. Byrne, E. Hemberg, A. Brabazon, M. O'Neill, A local search interface for interactive evolutionary architectural design, in *Proceedings of the 1st International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design, EvoMUSART 2012*, Malaga, Spain, 11–13 Apr. 2012, ed. by P. Machado, J. Romero, A. Carballal. Lecture Notes in Computer Science, vol. 7247 (Springer, Berlin, 2012), pp. 23–34
7. N. Chomsky, Three models for the description of language. IRE Trans. Inf. Theory **2**(3), (1956)
8. N. Chomsky, On certain formal properties of grammars. Inf. Control (2), 137–167 (1959)
9. N. Chomsky, M. Schutzenberger, On certain formal properties of grammars, in *Computer Programming and Formal Languages* (North Holland, Amsterdam, 1963)
10. R. Cleary, Extending grammatical evolution with attribute grammars: an application to knapsack problems. Master of science in computer science, University of Limerick, Ireland, 2005
11. R. Cleary, M. O'Neill, An attribute grammar decoder for the 01 multiconstrained Knapsack problem, in *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2005*, Lausanne, Switzerland, 30 Mar.–1 Apr. 2005, ed. by G.R. Raidl, J. Gottlieb. Lecture Notes in Computer Science, vol. 3448 (Springer, Berlin, 2005), pp. 34–45
12. B. Cody-Kenny, M. Fenton, A. Ronayne, E. Considine, T. McGuire, M. O'Neill, A search for improved performance in regular expressions, in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17*, Berlin, Germany, 15–19 July 2017 (ACM, New York, 2017), pp. 1280–1287
13. I. Dempsey, M. O'Neill, A. Brabazon, *Foundations in Grammatical Evolution for Dynamic Environments*. Studies in Computational Intelligence, vol. 194 (Springer, Berlin, 2009)
14. S. Donne, M. Nicolau, C. Bean, M. O'Neill, Wave height quantification using land based seismic data with grammatical evolution, in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, Beijing, China, 6–11 July 2014, ed. by C.A. Coello Coello, pp. 2909–2916
15. J.A. Driscoll, B. Worzel, D. MacLean, Classification of gene expression data with genetic programming, in *Genetic Programming Theory and Practice*, ed. by R.L. Riolo, B. Worzel, chap. 3 (Kluwer, Boston, 2003), pp. 25–42
16. D. Fagan, Analysing the genotype-phenotype map in grammatical evolution. Ph.D. Thesis, University College Dublin, Ireland, 30 Oct. 2013
17. D. Fagan, E. Hemberg, M. O'Neill, S. McGarraghy, Fitness reactive mutation in grammatical evolution, in *18th International Conference on Soft Computing, MENDEL 2012*, Brno, Czech Republic, 27–29 June 2012, ed. by R. Matousek (Brno University of Technology, Brno, 2012), pp. 144–149
18. D. Fagan, M. Fenton, M. O'Neill, Exploring position independent initialisation in grammatical evolution, in *Proceedings of 2016 IEEE Congress on Evolutionary Computation (CEC 2016)*, Vancouver, 24–29 July 2016, ed. by Y.-S. Ong (IEEE Press, New York, 2016), pp. 5060–5067

19. M. Fenton, D. Lynch, S. Kucera, H. Claussen, M. O'Neill, Multilayer optimization of heterogeneous networks using grammatical genetic programming. IEEE Trans. Cybern. **47**(9), 2938–2950 (2018)
20. M. Fenton, D. Lynch, S. Kucera, H. Claussen, M. O'Neill, Multilayer optimization of heterogeneous networks using grammatical genetic programming, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '17*, Berlin, Germany, 15–19 July 2017 (ACM, New York, 2017), pp. 3–4
21. M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, E. Hemberg, M. O'Neill, Ponyge2: Grammatical evolution in python, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO'17*, Berlin, Germany, 15–19 July 2017 (ACM, New York, 2017), pp. 1194–1201
22. M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, M. O'Neill, E. Hemberg, Ponyge2: grammatical evolution in python. arXiv, 26 Apr. 2017
23. C. Ferreira, Gene expression programming and the automatic evolution of computer programs, in *Recent Developments in Biologically Inspired Computing*, ed. by L.N. de Castro, F.J. Von Zuben, chap. 6 (Idea Group Publishing, Boulder, 2004), pp. 82–103
24. D. Gavrilis, I.G. Tsoulos, E. Dermatas, Selecting and constructing features using grammatical evolution. Pattern Recogn. Lett. **29**(9), 1358–1365 (2008)
25. R.T.R. Harper, Enhancing grammatical evolution. Ph.D. Thesis, School of Computer Science and Engineering, The University of New South Wales, Sydney 2052, Australia, 2009
26. R. Harper, Ge, explosive grammars and the lasting legacy of bad initialisation, in *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 18–23 July 2010 (IEEE Press, New York, 2010)
27. R. Harper, A. Blair, A structure preserving crossover in grammatical evolution, in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, Edinburgh, UK, 2–5 Sept. 2005, ed. by D. Corne, Z. Michalewicz, M. Dorigo, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, T. K. Chen, G. Raidl, A. Zalzala, S. Lucas, B. Paechter, J. Willies, J. J. M. Guervos, E. Eberbach, B. McKay, A. Channon, A. Tiwari, L. G. Volkert, D. Ashlock, M. Schoenauer, vol. 3 (IEEE Press, New York, 2005), pp. 2537–2544
28. R. Harper, A. Blair, A self-selecting crossover operator, in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, Vancouver, 6–21 July 2006, ed. by G.G. Yen, L. Wang, P. Bonissone, S.M. Lucas (IEEE Press, New York, 2006), pp. 5569–5576
29. R. Harper, A. Blair, Dynamically defined functions in grammatical evolution, in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, Vancouver (IEEE Press, 2006), pp. 9188–9195. https://doi.org/10.1109/CEC.2006.1688638
30. M. Hemberg, U.-M. O'Reilly, P. Nordin, GENR8 - a design tool for surface generation, in *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, San Francisco, California, USA, 9–11 July 2001, ed. by E.D. Goodman, pp. 160–167
31. E. Hemberg, N. McPhee, M. O'Neill, A. Brabazon, Pre-, in- and postfix grammars for symbolic regression in grammatical evolution, in *IEEE Workshop and Summer School on Evolutionary Computing*, University of Ulster, Derry, Northern Ireland, 18–22 Aug. 2008, ed. by T.M. McGinnity, pp. 18–22
32. E. Hemberg, L. Ho, M. O'Neill, H. Claussen, A comparison of grammatical genetic programming grammars for controlling femtocell network coverage. Genet. Program. Evolvable Mach. **14**(1), 65–93 (2013)
33. E. Hemberg, C. Gilligan, M. O'Neill, A. Brabazon, A grammatical genetic programming approach to modularity in genetic algorithms, in *Proceedings of the 10th European Conference on Genetic Programming*, Valencia, Spain, ed. by M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A.I. Esparcia-Alcázar. Lecture Notes in Computer Science, vol. 4445 (Springer, Berlin, 2007), pp. 1–11. https://doi.org/10.1007/978-3-540-71605-1_1
34. J. Hugosson, E. Hemberg, A. Brabazon, M. O'Neill, Genotype representations in grammatical evolution. Appl. Soft Comput. **10**(1), 36–43 (2010) https://doi.org/10.1016/j.asoc.2009.05.003
35. M.R. Karim, C. Ryan, A new approach to solving 0-1 multiconstraint knapsack problems using attribute grammar with lookahead, in *Proceedings of the 14th European Conference*

on Genetic Programming, EuroGP 2011, Turin, Italy, 27–29 Apr. 2011, ed. by S. Silva, J.A. Foster, M. Nicolau, M. Giacobini, P. Machado. Lecture Notes in Computer Science, vol. 6621 (Springer, Berlin, 2011), pp. 250–261
36. M.R. Karim, C. Ryan, On improving grammatical evolution performance in symbolic regression with attribute grammar, in *GECCO Comp'14: Proceedings of the 2014 Conference Companion on Genetic and Evolutionary Computation Companion*, Vancouver, BC, Canada, 12–16 July 2014, ed. by C. Igel, D. V. Arnold, C. Gagne, E. Popovici, A. Auger, J. Bacardit, D. Brockhoff, S. Cagnoni, K. Deb, B. Doerr, J. Foster, T. Glasmachers, E. Hart, M.I. Heywood, H. Iba, C. Jacob, T. Jansen, Y. Jin, M. Kessentini, J.D. Knowles, W. B. Langdon, P. Larranaga, S. Luke, G. Luque, J.A.W. McCall, M.A. Montes de Oca, A. Motsinger-Reif, Y. S. Ong, M. Palmer, K.E. Parsopoulos, G. Raidl, S. Risi, G. Ruhe, T. Schaul, T. Schmickl, B. Sendhoff, K.O. Stanley, T. Stuetzle, D. Thierens, J. Togelius, C. Witt, C. Zarges (ACM, New York, 2014), pp. 139–140
37. M. Keijzer, Scientific discovery using genetic programming. Ph.D. Thesis, Danish Technical University, IMM, Institute for Mathematical Modelling, Digital Signal Processing group, DK-2800 Lyngby, Denmark, Mar. 2002
38. M. Keijzer, V. Babovic, Dimensionally aware genetic programming, in *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida, USA, 13–17 July 1999, ed. by W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith, vol. 2 (Morgan Kaufmann, Burlington, 1999), pp. 1069–1076
39. M. Keijzer, V. Babovic, C. Ryan, M. O'Neill, M. Cattolico, Adaptive logic programming, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, California, USA, 7–11 July 2001, ed. by L. Spector, E. D. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, E. Burke (Morgan Kaufmann, Burlington, 2001), pp. 42–49
40. M. Keijzer, M. O'Neill, C. Ryan, M. Cattolico, Grammatical evolution rules: The mod and the bucket rule, in *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, Kinsale, Ireland, 3–5 Apr. 2002, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi. Lecture Notes in Computer Science, vol. 2278 (Springer, Berlin, 2002), pp. 123–130
41. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, MA, USA, 1992)
42. J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press, Cambridge MA, 1994)
43. J.R. Koza, D. Andre, F.H. Bennett III, M. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving* (Morgan Kaufmann, San Francisco, 1999)
44. J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence* (Kluwer Academic, Norwell, 2003)
45. W.B. Langdon, R. Poli, *Foundations of Genetic Programming* (Springer, Berlin, 2002)
46. N. Lourenco, F.B. Pereira, E. Costa, SGE: a structured representation for grammatical evolution, in *Artificial Evolution*, Lyon, France, 26–28 Oct. 2015, ed. by S. Bonnevay, P. Legrand, N. Monmarche, E. Lutton, M. Schoenauer, *Lecture Notes in Computer Science*, vol. 9554 (Springer, Cham, 2015), pp. 136–148
47. J. McDermott, Graph grammars for evolutionary 3D design. Genet. Program. Evolvable Mach. **14**(3), 369–393 (2013). Special issue on biologically inspired music, sound, art and design.
48. J. McDermott, M. O'Neill, A. Brabazon, Interactive interpolating crossover in grammatical evolution, in *2010 IEEE World Congress on Computational Intelligence*, Barcelona, Spain, 18–23 July 2010 (IEEE Computational Intelligence Society/IEEE Press, New York, 2010), pp. 3018–3025
49. R.I. McKay, N.X. Hoai, P.A. Whigham, Y. Shan, M. O'Neill, Grammar-based genetic programming: a survey. Genet. Program. Evolvable Mach. **11**(3/4), 365–396 (2010). Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines
50. D.J. Montana, Strongly typed genetic programming. Evol. Comput. **3**(2), 199–230 (1995)

51. J.H. Moore, J.S. Parker, L.W. Hahn, Symbolic discriminant analysis for mining gene expression patterns, in *Machine Learning: EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5–7, 2001, Proceedings*, 2001, pp. 372–381
52. A. Moraglio, J. McDermott, M. O'Neill, Geometric semantic grammatical evolution, in *Semantic Methods in Genetic Programming*, Ljubljana, Slovenia, 13 Sept. 2014, ed. by C. Johnson, K. Krawiec, A. Moraglio, M. O'Neill. Workshop at Parallel Problem Solving from Nature 2014 conference.
53. J.E. Murphy, Applications of evolutionary computation to quadrupedal animal animation. Ph.D. Thesis, School of Computer Science and Informatics, University College Dublin, Ireland, Mar. 2011
54. E. Murphy, An exploration of tree-adjoining grammars for grammatical evolution. Ph.D. Thesis, University College Dublin, Ireland, 6 Dec. 2014
55. J. Murphy, M. O'Neill, H. Carr, Exploring grammatical evolution for horse gait optimisation, in *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, Tuebingen, Apr. 15–17 2009, ed. by L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, M. Ebner. Lecture Notes in Computer Science, vol. 5481 (Springer, Berlin, 2009), pp. 183–194
56. E. Murphy, M. O'Neill, E. Galvan-Lopez, A. Brabazon, Tree-adjunct grammatical evolution, in *2010 IEEE World Congress on Computational Intelligence*, Barcelona, Spain, 18–23 July 2010 (IEEE Computational Intelligence Society/IEEE Press, New York, 2010), pp. 4449–4456
57. J.E. Murphy, H. Carr, M. O'Neill, Animating horse gaits and transitions, in *Eighth Theory and Practice of Computer Graphics TPCG 2010*, Sheffield, UK, 6–8 Sept. 2010, ed. by J. Collomosse, I. Grimstead. Eurographics
58. E. Murphy, M. Nicolau, E. Hemberg, M. O'Neill, A. Brabazon, Differential gene expression with tree-adjunct grammars, in *Parallel Problem Solving from Nature, PPSN XII (part 1)*, Taormina, Italy, Sept. 1–5 2012, ed. by C.A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone. *Lecture Notes in Computer Science*, vol. 7491 (Springer, Berlin, 2012), pp. 377–386
59. M. Nicolau, Understanding grammatical evolution: initialisation. Genet. Program. Evolvable Mach. **18**(4), 467–507 (2017)
60. M. Nicolau, M. Saunders, M. O'Neill, B. Osborne, A. Brabazon, Evolving interpolating models of net ecosystem co2 exchange using grammatical evolution, in *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, ed. by A. Moraglio, S. Silva, K. Krawiec, P. Machado, C. Cotta. Lecture Notes in Computer Science, vol. 7244, Malaga, Spain, 11–13 Apr. 2012 (Springer, Berlin, 2012), pp. 134–145
61. M. O'Driscoll, S. McKenna, J.J. Collins, Synthesising edge detectors with grammatical evolution, in *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, New York, 8 July 2002, ed. by A.M. Barry (AAAI, Menlo Park, 2002), pp. 137–140
62. C. Oesch, D. Maringer, A neutral mutation operator in grammatical evolution, in *IEEE Conference on Intelligent Systems (1)*, ed. by P.P. Angelov, K.T. Atanassov, L. Doukovska, M. Hadjiski, V.S. Jotsov, J. Kacprzyk, N. Kasabov, S. Sotirov, E. Szmidt, S. Zadrozny. Advances in Intelligent Systems and Computing, vol. 322 (Springer, Berlin, 2014), pp. 439–449
63. M. O'Neill, A. Brabazon, mGGA: the meta-grammar genetic algorithm, in *Proceedings of the 8th European Conference on Genetic Programming*, Lausanne, Switzerland, 30 Mar.–1 Apr. 2005, ed. by M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, M. Tomassini. Lecture Notes in Computer Science, vol. 3447 (Springer, Berlin, 2005), pp. 311–320
64. M. O'Neill, A. Brabazon, Grammatical differential evolution, in *Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006*, Las Vegas, Nevada, USA, June 26–29 2006, ed. by H.R. Arabnia, vol. 1 (CSREA Press, Athens, 2006), pp. 231–236
65. M. O'Neill, A. Brabazon, Grammatical swarm: the generation of programs by social programming. Nat. Comput. **5**(4), 443–462 (2006)

66. M. O'Neill, A. Brabazon, Evolving a logo design using Lindenmayer systems, postscript and grammatical evolution, in *2008 IEEE World Congress on Computational Intelligence*, Hong Kong, 1–6 June 2008, ed. by J. Wang (IEEE Computational Intelligence Society/IEEE Press, New York, 2008), pp. 3788–3794
67. M. O'Neill, C. Ryan, Automatic generation of caching algorithms, in *Evolutionary Algorithms in Engineering and Computer Science*, Jyväskylä, Finland, 30 May–3 June 1999, ed. by K. Miettinen, M.M. Mäkelä, P. Neittaanmäki, J. Periaux (Wiley, New York, 1999), pp. 127–134
68. M. O'Neill, C. Ryan, Evolving multi-line compilable C programs, in *Genetic Programming, Proceedings of EuroGP'99*, Goteborg, Sweden, 26–27 May 1999, ed. by R. Poli, P. Nordin, W.B. Langdon, T.C. Fogarty. Lecture Notes in Computer Science, vol. 1598 (Springer, Berlin, 1999), pp. 83–92
69. M. O'Neill, C. Ryan, Crossover in grammatical evolution: a smooth operator? in R. Poli, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty, *Genetic Programming, Proceedings of EuroGP'2000*, Edinburgh, 15–16 Apr. 2000. Lecture Notes in Computer Science, vol. 1802 (Springer, Berlin, 2000), pp. 149–162
70. M. O'Neill, C. Ryan, Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code, in *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, Coimbra, Portugal, 5–7 Apr. 2004, ed. by M. Keijzer, U.-M. O'Reilly, S.M. Lucas, E. Costa, T. Soule. Lecture Notes in Computer Science, vol. 3003 (Springer, Berlin, 2004), pp. 138–149
71. M. O'Neill, J.J. Collins, C. Ryan, Automatic generation of robot behaviours using grammatical evolution, in *Proceedings of the Fifth International Symposium on Artificial Life and Robotics*, Oita, Japan, 26–28 Jan. 2000, ed. by M. Sugisaka, H. Tanaka (2000), pp. 351–354
72. M. O'Neill, C. Ryan, M. Keijzer, M. Cattolico, Crossover in grammatical evolution: the search continues, in *Genetic Programming, Proceedings of EuroGP'2001*, Lake Como, Italy, 18–20 Apr. 2001, ed. by J.F. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A.G.B. Tettamanzi, W.B. Langdon. Lecture Notes in Computer Science, vol. 2038 (Springer, Berlin, 2001), pp. 337–347
73. M. O'Neill, C. Ryan, M. Keijzer, M. Cattolico, Crossover in grammatical evolution. Genet. Program. Evolvable Mach. **4**(1), 67–93 (2003)
74. M. O'Neill, A. Brabazon, C. Adley, The automatic generation of programs for classification problems with grammatical swarm, in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, Portland, Oregon, 20–23 June 2004 (IEEE Press, New York, 2004), pp. 104–110
75. M. O'Neill, R. Cleary, N. Nikolov, Solving Knapsak problems with attribute grammars, in *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, 26–30 June 2004, ed. by R. Poli, S. Cagnoni, M. Keijzer, E. Costa, F. Pereira, G. Raidl, S.C. Upton, D. Goldberg, H. Lipson, E. de Jong, J. Koza, H. Suzuki, H. Sawai, I. Parmee, M. Pelikan, K. Sastry, D. Thierens, W. Stolzmann, P.L. Lanzi, S.W. Wilson, M. O'Neill, C. Ryan, T. Yu, J.F. Miller, I. Garibay, G. Holifield, A.S. Wu, T. Riopka, M.M. Meysenburg, A.W. Wright, N. Richter, J.H. Moore, M.D. Ritchie, L. Davis, R. Roy, M. Jakiela (2004)
76. M. O'Neill, A. Brabazon, E. Hemberg, Subtree deactivation control with grammatical genetic programming in dynamic environments, in *2008 IEEE World Congress on Computational Intelligence*, Hong Kong, 1–6 June 2008, ed. by J. Wang (IEEE Computational Intelligence Society/IEEE Press, New York, 2008), pp. 3768–3774
77. M. O'Neill, J. McDermott, J.M. Swafford, J. Byrne, E. Hemberg, A. Brabazon, E. Shotton, C. McNally, M. Hemberg, Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. Int. J. Des. Eng. **3**(1), 4–24 (2010)
78. M. O'Neill, L. Vanneschi, S. Gustafson, W. Banzhaf, Open issues in genetic programming. Genet. Program. Evolvable Mach. **11**(3/4), 339–363 (2010). Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.
79. M. O'Neill, M. Nicolau, A. Brabazon, Dynamic environments can speed up evolution with genetic programming, in *GECCO'11: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, Dublin, Ireland, 12–16 July 2011, ed. by N. Krasnogor, P. L. Lanzi, A. Engelbrecht, D. Pelta, C. Gershenson, G. Squillero, A. Freitas, M. Ritchie, M. Preuss, C. Gagne, Y.S. Ong, G. Raidl, M. Gallager, J. Lozano, C. Coello-Coello, D.L. Silva, N. Hansen, S. Meyer-Nieberg, J. Smith, G. Eiben, E. Bernado-Mansilla,

W. Browne, L. Spector, T. Yu, J. Clune, G. Hornby, M.-L. Wong, P. Collet, S. Gustafson, J.-P. Watson, M. Sipper, S. Poulding, G. Ochoa, M. Schoenauer, C. Witt, A. Auger (ACM, New York, 2011), pp. 191–192
80. M. O'Neill, M. Nicolau, A. Agapitos, Experiments in program synthesis with grammatical evolution: a focus on integer sorting, in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, Beijing, China, 6–11 July 2014, ed. by C.A. Coello Coello (2014), pp. 1504–1511
81. M. O'Neill, A. Brabazon, D. Fagan, An exploration of grammatical encodings to model six nations rugby match outcomes, in *Proceedings of 2016 IEEE Congress on Evolutionary Computation (CEC 2016)*, Vancouver, 24–29 July 2016, ed. by Y.-S. Ong (IEEE Press, New York, 2016), pp. 4429–4436
82. M. O'Neill, C. Ryan, Grammar based function definition in grammatical evolution, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Las Vegas, NV, ed. by D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, H.-G. Beyer (Morgan Kaufmann, San Francisco, 2000), pp. 485–490
83. U.-M. O'Reilly, M. Hemberg, Integrating generative growth and evolutionary computation for form exploration. Genet. Program. Evolvable Mach. **8**(2), 163–186 (2007). Special issue on developmental systems.
84. A. Ortega, M. de la Cruz, M. Alfonseca, Christiansen grammar evolution: grammatical evolution with semantics. IEEE Trans. Evol. Comput. **11**(1), 77–90 (2007)
85. J. O'Sullivan, C. Ryan, An investigation into the use of different search strategies with grammatical evolution, in *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, Kinsale, Ireland, 3–5 Apr. 2002, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi. Lecture Notes in Computer Science, vol. 2278 (Springer, Berlin, 2002), pp. 268–277
86. C. Ryan, R.M.A. Azad, Sensible initialisation in grammatical evolution, in *GECCO 2003: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, Chicago, 11 July 2003, ed. by A.M. Barry (AAAI, Menlo Park, 2003), pp. 142–145
87. C. Ryan, M. O'Neill, How to do anything with grammars, in *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, New York, 8 July 2002, ed. by A.M. Barry (AAAI, Menlo Park, 2002), pp. 116–119
88. C. Ryan, M. Nicolau, M. O'Neill, Genetic algorithms using grammatical evolution, in *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, Kinsale, Ireland, 3–5 Apr. 2002, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi. Lecture Notes in Computer Science, vol. 2278 (Springer, Berlin, 2002), pp. 278–287
89. C. Ryan, M. Keijzer, M. Nicolau, On the avoidance of fruitless wraps in grammatical evolution, in *Genetic and Evolutionary Computation – GECCO-2003*, Chicago, 12–16 July 2003, ed. by E. Cantú-Paz, J.A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M.A. Potter, A.C. Schultz, K. Dowsland, N. Jonoska, J. Miller. Lecture Notes in Computer Science, vol. 2724 (Springer, Berlin, 2003), pp. 1752–1763
90. J.M. Swafford, M. O'Neill, M. Nicolau, A. Brabazon, Exploring grammatical modification with modules in grammatical evolution, in *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, Turin, Italy, ed. S. Silva, J.A. Foster, M. Nicolau, M. Giacobini, P. Machado. Lecture Notes in Computer Science, vol. 6621. (Springer, Berlin, 2011), pp. 310–321. https://doi.org/10.1007/978-3-642-20407-4_27
91. J.M. Swafford, E. Hemberg, M. O'Neill, A. Brabazon, Analyzing module usage in grammatical evolution, in *Parallel Problem Solving from Nature, PPSN XII (part 1)*, Taormina, Italy, ed. by C.A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone. Lecture Notes in Computer Science, vol. 7491 (Springer, Berlin, 2012), pp. 347–356. https://doi.org/10.1007/978-3-642-32937-1_35
92. J.M. Swafford, E. Hemberg, M. O'Neill, M. Nicolau, A. Brabazon, A non-destructive grammar modification approach to modularity in grammatical evolution, in *Gecco'11: Proceedings of*

the 13th Annual Conference on Genetic and Evolutionary Computation, Dublin, Ireland, ed. by N. Krasnogor, P.L. Lanzi, A. Engelbrecht, D. Pelta, C. Gershenson, G. Squillero, A. Freitas, M, Ritchie, M. Preuss, C. Gagne, Y.S. Ong, G. Raidl, M. Gallager, J. Lozano, C. Coello-Coello, D.L. Silva, N. Hansen, S. Meyer-Nieberg, J. Smith, G. Eiben, E. Bernado-Mansilla, W. Browne, L. Spector, T. Yu, J. Clune, G. Hornby, M.-L. Wong, P. Collet, S. Gustafson, J.-P. Watson, M. Sipper, S. Poulding, G. Ochoa, M. Schoenauer, C. Witt, A. Auger (ACM, New York, 2011), pp. 1411–1418. https://doi.org/10.1145/2001576.2001766

93. J. Swafford, M. Nicolau, E. Hemberg, M. O'Neill, A. Brabazon, Comparing methods for module identification in grammatical evolution, in *GECCO'12: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, Philadelphia, PA, ed. by T. Soule, A. Auger, J. Moore, D. Pelta, C. Solnon, M. Preuss, A. Dorin, Y.-S. Ong, C. Blum, D.L. Silva, F. Neumann, T. Yu, A. Ekart, W. Browne, T. Kovacs, M.-L. Wong, C. Pizzuti, J. Rowe, T. Friedrich, G. Squillero, N. Bredeche, S.L. Smith, A. Motsinger-Reif, J. Lozano, M. Pelikan, S. Meyer-Nienberg, C. Igel, G. Hornby, R. Doursat, S. Gustafson, G. Olague, S. Yoo, J. Clark, G. Ochoa, G. Pappa, F. Lobo, D. Tauritz, J. Branke, K. Deb (ACM, New York, 2012), pp. 823–830. https://doi.org/10.1145/2330163.2330277
94. A. Thompson, Silicon evolution, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, 28–31 July 1996, ed. by J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (MIT Press, Cambridge, 1996), pp. 444–452
95. P.A. Whigham, Grammatically-based genetic programming, in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, Tahoe City, California, USA, 9 July 1995, ed. by J.P. Rosca (1995), pp. 33–41
96. P.A. Whigham, A schema theorem for context-free grammars, in *1995 IEEE Conference on Evolutionary Computation*, Perth, Australia, 29 Nov.–1 Dec. 1995, vol. 1 (IEEE Press, New York, 1995), pp. 178–181
97. P.A. Whigham, Grammatical bias for evolutionary learning. Ph.D. Thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 14 Oct. 1996
98. P.A. Whigham, Search bias, language bias, and genetic programming, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, 28–31 July 1996, ed. by J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (MIT Press, Cambridge, 1996), pp. 230–237