Genet Program Evolvable Mach (2010) 11:365–396
DOI 10.1007/s10710-010-9109-y

CONTRIBUTED ARTICLE

Grammar-based Genetic Programming: a survey

Robert I. McKay · Nguyen Xuan Hoai · Peter Alexander Whigham · Yin Shan · Michael O'Neill

Received: 23 November 2009 / Revised: 10 April 2010 / Published online: 1 May 2010

Abstract Grammar formalisms are one of the key representation structures in Computer Science. So it is not surprising that they have also become important as a method for formalizing constraints in Genetic Programming (GP). Practical grammar-based GP systems first appeared in the mid 1990s, and have subsequently become an important strand in GP research and applications. We trace their subsequent rise, surveying the various grammar-based formalisms that have been used in GP and discussing the contributions they have made to the progress of GP. We illustrate these contributions with a range of applications of grammar-based GP, showing how grammar formalisms contributed to the solutions of these problems. We briefly discuss the likely future development of grammar-based GP systems, and conclude with a brief summary of the field.

Keywords Genetic programming · Evolutionary computation · Grammar · Context free · Regular · Tree adjoining

R. I. McKay (✉)
Structural Complexity Lab, School of Computer Science and Engineering,
Seoul National University, Seoul, Korea
e-mail: rimsnucse@gmail.com

N. X. Hoai
Department of Computer Science, Le Quy Don University, Hanoi, Vietnam

P. A. Whigham
Department of Information Science, University of Otago, Dunedin, NZ, New Zealand

Y. Shan
Medicare Australia, Canberra, Australia

M. O'Neill
Complex and Adaptive Systems Lab, School of Computer Science and Informatics,
University College Dublin, Dublin, Ireland

1 Introduction

Various algorithms that might loosely be called Genetic Programming (GP) were described in the 1980s and even earlier (the closest to modern GP being the work of Cramer [10] and of Hicklin [32]). However it was the research of Koza [47] which clearly defined the field and established it as an important sub-field of evolutionary computation. In the remainder of this paper, we refer to this archetypical—and still widely used—form of GP as "standard" GP.

Grammars are core representation structures in Computer Science. They are widely used to represent restrictions on general domains, limiting the expressions that may be used. They can be used to define the legal expressions of a computer language, to impose type restrictions, or to describe constraints on interactions within systems. So it is not surprising that grammars have played an important role in the development of GP. They have been used in GP almost from its inception. Indeed the idea was implicit in Ch. 19 of Koza's first book [47]. At first, grammar-based approaches formed a small segment of the GP world, but their role has expanded to the point where Grammatical Evolution (GE [67, 68]) is now one of the most widely applied GP methods.

We aim to give a flavour of the history and applications of grammar-based GP, from the emergence of the first fully-fledged systems in the mid 1990s (throughout the paper, to speak generically of the field, we will use the term Grammar Guided Genetic Programming—GGGP). We then examine the pros and cons of GGGP to try to understand why it has become so influential, and delineate some of the issues we think will be important in the future. The field continues to develop, so we have arbitrarily limited the survey up to the end of 2008, as a practical dividing line.

In Sect. 2 we introduce the most straightforward version, tree-structured Chomsky grammar GP. In Sect. 3, we examine some linearised versions based on these grammars. We follow this, in Sect. 4, with some alternative approaches that use different or extended grammar models. Section 5 concludes the historical review with a consideration of other population-based stochastic algorithms using grammars to represent GP problem spaces. Having considered the different forms of grammar-based GP, we then look, in Sect. 6, at the range of areas in which GGGP techniques have been applied. We explain the role of GGGP in Sect. 7, and examine its advantages (and disadvantages) relative to other GP methods in Sect. 8. We attempt, in Sect. 9, to forecast the important future directions of the field, and present our conclusions in Sect. 10.

1.1 Terminology

In the following sections the term *genotype* refers to the structure operated on by genetic operators such as crossover and mutation. Hence for tree based GGGP the genotype is a derivation tree in the language defined by a grammar, while for a linear representation such as GE the genotype is a linear string which is then decoded into a derivation tree. The term *phenotype* refers to the structure that is directly executed to produce the behaviour of an individual. Hence for both tree based GGGP and GE, the phenotype refers to the expression tree (GP-style tree)

which is directly evaluated. The grammar defines the interpretation of the genotype space, and in tree-based systems directly defines the set of valid derivation trees (i.e. genotypes). In some representations, such as GE and TAG (described below), the genotype is first decoded to a context-free derivation tree, before further transformation to an expression tree. In these cases, we refer to this intermediate stage as an "intermediate state".

2 Tree based grammar guided Genetic Programming

The first fully-fledged GGGP systems were independently implemented at about the same time by three different research groups. Whigham [102] proposed the CFG-GP system, in which a context-free grammar (CFG) was used to generate the population, consisting of derivation trees of the CFG. Geyer-Schulz [22] derived his very similar GGGP system for learning knowledge rules for expert systems; it differs mainly in the algorithm used to initialise the population [3]. Wong and Leung [108] proposed the LOGENPRO system, which uses PROLOG Definite Clause Grammars (DCGs) to generate (logic) programs to learn first order relations. DCGs are somewhat more expressive than CFGs, being able to generate some context-sensitive languages, which can be important for some applications. This is the primary difference between the systems; in other respects, LOGENPRO and CFG-GP are very similar.

GGGP example We illustrate GGGP using a generalisation of Dawkins' example [14] for explaining evolution: hidden string matching. We use CFG-GP, but very little change would be needed for LOGENPRO or Geyer-Schulz's system. In Genetic Algorithm string matching, the requirement is to match a hidden string, and the (minimising) fitness function is the number of mismatches. In our example, the requirement is to match a hidden sentence—perhaps "the cat sat on the mat". The (minimising) fitness function is the number of edit operations required to match the words of the sentences, and the GGGP system is only permitted to evaluate syntactically valid sentences. This would be a difficult problem to encode in standard GP.

The five basic components of a GGGP system are as follows:

1. **Program representation** Each program is a derivation tree generated by a grammar G; the grammar defines a language L whose terms are expressions corresponding to those used in "standard" GP. In our example, G is the fragment of English grammar shown in Table 1.[1]
2. **Population initialisation** Whigham [102] proposed a simple algorithm to generate random derivation trees up to a depth bound, based on a procedure for labeling the production rules with the minimum tree depth required to produce a string of terminals. Bohm and Geyer-Schulz [3] derived an algorithm for initialising the population based on the derivation-step-uniform distribution. The

[1] Reproduced under the Creative Commons Licence; available from https://sc.snu.ac.kr/sclab/doku.php?id=commons.

Table 1 English grammar fragment

Sent → Sub Pred	PP → Prep NP	Prep → ''on''|''under''
Sub → NP	Adjs → Adj Adjs	Noun → ''cat''|''dog''
Pred → Verb PP	Adjs → Adj	|''floor'' |''mat''
NP → Det Noun	Verb → ''sat''|''stood''	Adj → ''big''|''small''
NP → Det Adjs Noun	Det → ''a''|''the''	|''red'' |''black''

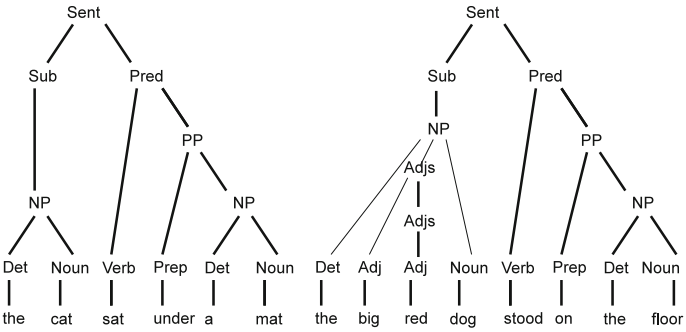

Fig. 1 Typical parse trees from example grammar

initialisation procedure for LOGENPRO was based on Prolog grammar generation. In our example, we might initially generate parse trees for sentences such as "the cat sat under a mat" or "the big red dog stood on the floor", as in Fig. 1.

3. **Fitness evaluation** Fitness evaluation is carried out on the individuals by reading the expression tree from the leaves of the grammar derivation tree, then evaluating it as in standard GP. In our example, using edit distance, the distance from the first sentence in Fig. 1 to the target is 2 (replacing "under" with "on" and "a" with "the"), and from the second it is 5 (deleting "big" and "red", substituting "dog" with "cat", "stood" with "sat" and "floor" with "mat").
4. **Parameters** As in standard GP—population size, number of generations, maximal tree depth, and operator probabilities.
5. **Genetic operators** The genetic operators are the selection mechanism, reproduction, crossover, and mutation. Selection and reproduction are as in standard GP. Crossover and mutation are slight variants of those used in standard GP, as explained below.

   Crossover Internal nodes of the two derivation trees, labelled with the same nonterminal symbol of the grammar, are chosen at random and the two sub-derivation trees underneath them exchanged. In other words, the crossover acts the same as standard GP subtree crossover, with the additional constraint that crossover points are required to have the same grammar label. For example, we might select the leftmost NP of the left tree and the rightmost of the right tree as crossover points, as seen in Fig. 2.[1] The results of crossover are the child sentences

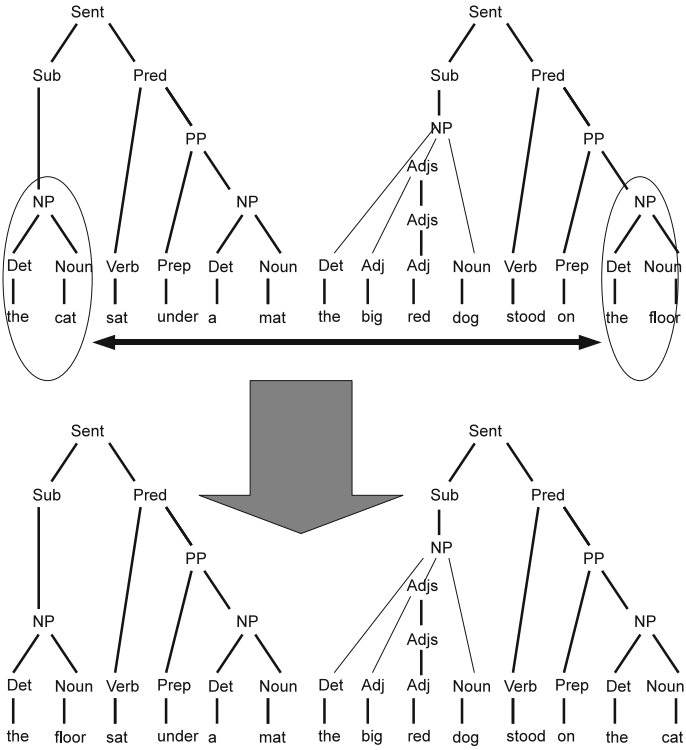

Fig. 2 Subtree crossover restricted by example grammar

"the floor sat under a mat" and "the big red dog stood on the cat".[2]

Mutation Selects an internal node at random. The subtree rooted at that node is deleted, and replaced by a new one, randomly generated according to the grammar, starting from the same nonterminal. For example, we might select the second of the previous child subtrees for mutation, and choose the first NP node in it as a mutation point, generating a new NP at this point, such as "the cat stood on the cat". This process is illustrated in Fig. 3.[1]

Subsequent to these systems, there have been a number of similar GGGP systems using derivation trees from a grammar as the representation for individuals. Gruau [25] presented some strong arguments for using context-free grammars to set language bias on individuals. His system is very similar to Whigham's CFG-GP, with differences mainly lying in some additional constraints imposed on using the grammar. The initial maximum individual size is constrained by an upper bound on recursive grammar applications. He introduces two additional data structures, lists and sets. The two structures support different crossover operators. These extensions,

[2] Recombinations in biological systems are usually homologous—the genetic materials are not exchanged completely randomly, but between genes with similar function [81]. By analogy, we term crossovers exchanging subtrees rooted at the same symbols "homologous crossovers".

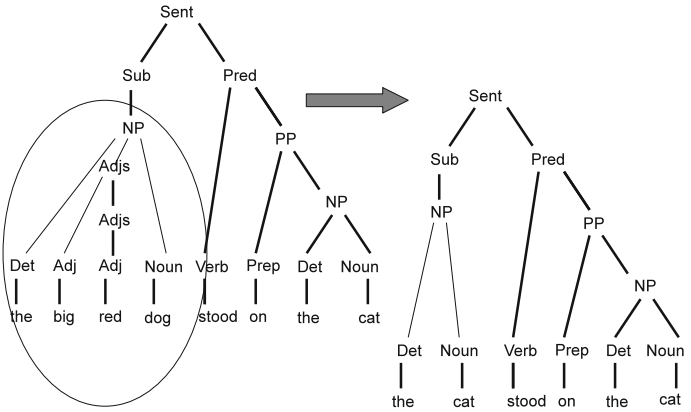

Fig. 3 Subtree mutation restricted by example grammar

introduced to allow specific restrictions in the language bias, complicated the model and may have been more readily introduced via a more expressive grammar.

Keijzer and Babovic [43] and Ratle and Sebag [77] introduced a grammar guided Genetic Programming system similar to the preceding systems, but with a different initialisation process, to solve some industrial problems. The resulting system was known as dimensionally-aware Genetic Programming.

Some recent variants are very similar to the above systems, but use different notations to represent CFGs. Tanev et al. [97] represent the CFG in Backus-Naur Form (BNF). MacCallum [51] introduced a Perl-based system, in which Perl tree notation is used to represent grammars equivalent to CFGs.

3 Linearised grammar guided Genetic Programming

Linear representations offer a number of seductive advantages over tree-based representations, especially the highly constrained tree-based representations arising from Chomsky grammars. Most obviously, linear representations permit the application of a vast background of both theory and practice from the much wider fields of evolutionary computation with fixed-length linear representations, notably Genetic Algorithms (GA) and Evolution Strategies (ES). Thus linear representations have become increasingly important in the non-grammar-based GP world, including such approaches as linear GP [35], machine-coded GP [61], stack-based GP [94] and Cartesian Genetic Programming [58]. Hence it is not surprising that there have been a number of approaches to linearising the representation of programs in GGGP. All use a genotype to phenotype mapping, where the genotype is a linear sequence, and an intermediate state is constructed in the form of a derivation tree of the grammar, which is then further decoded to an expression tree. The work of Johnson and Feyock [39] prefigured these approaches by several years, but was not widely available and so did not influence the subsequent development of the field.

In the early work of Keller and Banzhaf [46] a grammar was used during repair of the genotype-phenotype mapping to ensure the syntactic correctness of the phenotypic program. In this approach the genotype contained a series of codons (represented by a predetermined number of bits) to encode a symbol of the output language. The order of the codons determined the symbol order for the output program. However, it was only after the grammar-based repair operation that the syntactic correctness of the program could be guaranteed.

In Paterson and Livesey [75, 76] and Freeman [19], the genotype was a fixed string used to encode the indices for derivation rules in grammar G. In these methods, the translation from a genotype to an intermediate state is carried out from left to right, and the intermediate state (G derivation tree) is built correspondingly. At each step, if there is still an incomplete branch in the intermediate state marked by a nonterminal A, a gene (number of bits) is read and interpreted as a key, indicating which, among the rules in the rule set P of G having left hand side A, will be used to extend that branch of the intermediate state. If the intermediate state is completed while there are still unused genes in the genotype, they are ignored (considered as introns). In the event that the translation uses all the genes, but the intermediate state is still incomplete, some random or default sub-derivation trees are used to complete it. Because the genotype is now a linear string, the system can simply use any genetic operators from GAs.

The most widely used linear GGGP system is grammatical evolution (GE) [16, 68]. It is an extension of the GGGP systems with a linear representation described above. Three innovations were incorporated in GE: variable length (although these had been previously considered by Johnson and Feyock [39]), redundancy using the MOD mapping rule, and the wrapping operation. The chromosome in GE typically has variable length rather than being fixed, although fixed-length instances have been adopted [53, 66]. Each codon is typically an 8-bit binary number (integer codons are also widely adopted), which is used, as in the previous systems, to determine the rule for a nonterminal symbol when it is expanded. However the modulo operator is now used to map the number to the defined range of production rules from the given nonterminal. An illustration of the mapping process is provided in Fig. 4. The wrapping operation can be used when the translation from genotype to intermediate state has run out of genes while the intermediate state is still incomplete. The translation process re-uses the gene from left to right. If the number of wrappings exceeds a predetermined maximal bound, then the translation finishes and the (invalid) individual is assigned a very poor fitness. This introduces an upper bound on the size of any individual, since it is possible that a finite linear genome can map to an infinite number of production applications.

If we look again at the example grammar adopted in Fig. 4 we can immediately appreciate the potential benefits of adopting a grammar-based form of GP. Consider if we wanted to modify the form of the solutions generated by GE in this case. With some simple changes to the grammar the form of expression generated can be dramatically altered.

The example grammar provided in Table 2 would have the effect of changing the expressions in Fig. 4 from arithmetic combinations of coefficients and variables, to

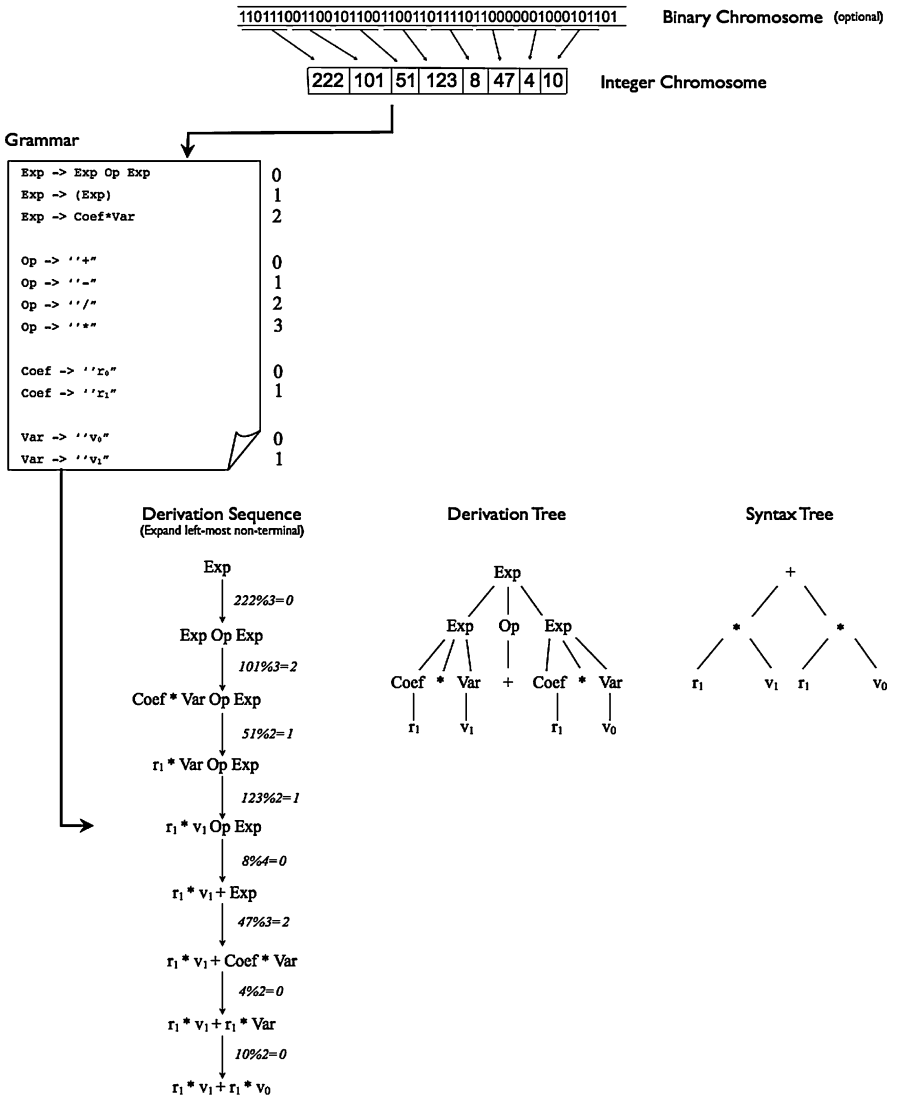

Fig. 4 An illustration of the Grammatical Evolution mapping from a linear binary (or integer) chromosome. The integer values are used in the mapping function to decide which production rule from the grammar to apply to the current non-terminal symbol. This results in the generation of a derivation sequence, which can be maintained as a derivation tree. The derivation tree is then reduced to the standard GP syntax tree. Note that search operators can be applied to both the linear chromosome and derivation tree

Table 2 Grammar for rational polynomials

Exp → Poly/Poly	Trm → Coef * Prod	Coef → ''x0''|''x1''
Poly → Trm	Prod → Var	Var → ''v0''|''v1''
Poly → Trm + Poly	Prod → Var * Prod	

more complex structures of rational polynomials. Recall that the original rules for <expr> simply generated arithmetic combinations of <coef>*<prod>.

Since it was proposed there has been a wide range of on-going research to develop, extend, and apply GE in many ways, including studying the effect of GE crossover on the phenotype [70], alternatives to the MOD rule in genotype-phenotype translation [45], an alternative mapping process that evolves the order in which nonterminals are expanded in addition to how they are expanded [65], different search strategies [63, 64, 73], new representations based on the GE representation aiming to reduce the effects of positional dependence [87], implementation of GAs through GE using an attribute grammar [86], the use of a meta-grammar with GE to improve its evolvability [69] and to implement a GA [29, 62]. Harper and Blair [27] showed how to use grammars to dynamically define functions, obviating the need for special-purpose mechanisms such as Automatically Defined Functions [48].

There are two key issues with the GE representation. Firstly, an apparently valid genotype may code for an infeasible (i.e. not fitness calculable) phenotype. Although the problem can be handled by assigning these individuals poor fitness values, it introduces a source of irregularity into the search space, and constitutes an obstacle for evolution and search on the GE representation if the proportion of genotypes coding for infeasible phenotypes is large. As such it is common to adopt repair during the genotype-phenotype mapping to ensure that all remaining nonterminals are mapped to terminals. Approaches include the use of default production rules which fire after the maximum number of wraps has been reached (originally adopted by Paterson and Livesey [76]), or the use of a dynamic grammar, which removes all nonterminal expanding rules from the grammar [30]. By using the equivalent of ramped-half-and-half initialisation with derivation trees (dubbed sensible initialisation), one can ensure the validity of at least the initial population [85].

Secondly, GE may not fulfil the locality principle of small changes in genotype resulting in small changes in phenotype [84]. Although it is easy to define GE operators that make small changes on the genotype, the resulting phenotype changes may be larger. A change at one position may change the expressiveness (coding or non-coding), or meaning (if there is more than one nonterminal in the grammar) of genes that lie after that position. In the extreme, this may change the corresponding phenotype from feasible to infeasible, or create a search process that is no better than random sampling. However, the extent of this effect depends on the relationship between the particular grammar, the genetic operators and the fitness function, and is clearly an issue with any complex genotype-phenotype mapping. Recent advances to address this issue have focused on the development of search operators towards the derivation trees that are produced during the genotype-phenotype mapping process. The result of these operations of crossover and mutation on the derivation tree (similar to those in tree-based GGGP) can be reverse-mapped back to the underlying genotypic binary strings [26]. Since the genotype remains the binary (or integer) string representation, these new operators may be readily combined with string-based operators, giving the best of both worlds in terms of available operators, though at the cost of losing some of the analytical understanding deriving from GA theory.

4 Alternative grammar representations

A number of alternative grammar-based representations have been used, in addition to those using Context Free Grammars. Most either extend the class of grammars into the domain of Context Sensitive Grammars, or incorporate semantic knowledge into the grammar representation, or both.

4.1 Semantic grammars

Hussain and Browse [37] noted that their NGAGE system for developmental generation of neural networks could also be used as a representation for general Genetic Programming, though it is not clear that they ever applied the system in this way. Vanyi and Zvada [100] implemented an attribute-grammar system to augment context-free GGGP. Their emphasis lay in the use of attributes to cache information required for evolutionary and evaluation operators, and thus speed up those operators; it does not appear that the system was ever used to incorporate problem-specific semantic information, or otherwise extended beyond CFG problem domains.

Bruhn and Geyer-Schulz [6] used CFGs with linear constraints for their representation; the constraints are a form of semantic attribute, making this the first use of attribute grammars to handle semantic information.

De la Cruz et al. [11] further considered evolution based on attribute grammars, considering semantic restrictions in the context of a symbolic regression problem, and demonstrating search speed-up when semantic constraints were incorporated. Their representation used a GE-like transformation of the attribute grammar search space.

Ortega et al. [72] investigated Christiansen grammars [7], which have equivalent expressive power to attribute grammars, but provide a more condensed representation for semantic constraints, and so may offer a cleaner search. They also use a GE-like linearisation. They demonstrated improved convergence on some artificial Boolean problems, though further clarification of the role of semantic constraint is needed. With Dalhoum [13], they achieved good results on location allocation problems.

Cleary and O'Neill adopted attribute grammars with GE in their application to knapsack problems [8]. The attribute grammars ensured that the constraints of the problem domain were adhered to as the solutions were being generated during the mapping process.

4.2 Logic grammars

Wong and Leung based their system, LOGENPRO [108], on logic grammars (Definite Clause Grammars—DCGs). These extend CFGs in two ways.

1. They support logical variables in grammar productions. Thus extended, DCGs provide limited context sensitivity. This was used in [108] to limit a search

space of recursive solutions; any recursion the system generated could be guaranteed to terminate.

2. They allow Prolog predicates to be intermixed with grammar productions; these additional predicates can incorporate semantic information, and thus give expressive power similar to (though in general greater than) attribute grammars. Wong and Leung used them in [108] to incorporate the semantics of Automatically Defined Functions (ADFs) into LOGENPRO.

In principle, they also permit the incorporation of semantic domain knowledge into LOGENPRO, though we are not aware of specific instances where this has been used. Consider, for example, the dimensionally-aware GP applications of Ratle and Sebag [77]; to incorporate dimensional knowledge into their system, Ratle and Sebag pre-process the domain knowledge into a huge CFG. This knowledge can be very succinctly expressed in a DCG (see Sect. 8 for details).

Ross [83] further extended this work, replacing DCGs with Definite Clause Translation Grammars, in which the semantics of the grammar are defined in a parallel structure to the syntax. This offers a key advantage in re-usability: the bulk of the problem semantics is written as annotations to the grammar, so the separate code to implement a fitness function is typically small—often less than ten lines of Prolog. Re-targeting to a new domain may involve very little effort—writing a grammar for the domain, together with semantic annotations.

Keijzer et al. [44] combined logic grammars with Grammatical Evolution in their ALP (Adaptive Logic Programming) system. ALP modifies a standard logic programming engine with an evolutionary-driven search engine.

4.3 Tree adjoining grammar representation

The grammar representations previously mentioned are all closely related to the Chomsky grammars of the 1950s. The differences between them relate essentially to the sub-class of formal languages that they can represent, and the means by which they may incorporate semantic constraints. However another form of representation, Tree Adjoining Grammars (TAGs) has become increasingly important in Natural Language Processing (NLP) since its introduction in the 1970s by Joshi et al. [40]. The aim of TAG representation is to more directly represent the structure of natural languages than is possible in Chomsky languages and, in particular, to represent the process by which natural language sentences can be built up from a relatively small set of basic linguistic units by inclusion of insertable sub-structures. Thus “The cat sat on the mat” becomes “The big black cat sat lazily on the comfortable mat which it had commandeered” by insertion of the elements “big”, “black”, “lazily”, “comfortable”, “which it had commandeered”. In CFG representation, the relationship between these two sentences can only be discerned by detailed analysis of their derivation trees; in TAG representation, the derivation tree of the latter simply extends the frontier of the former. To put it another way, the edit distance between the derivation trees of these closely related sentences is much smaller in TAG representation than in CFG representation.

The valuable properties which TAG representation introduces into NLP are arguably also of value in GP; in a series of papers, Nguyen has introduced two TAG-based GGGP systems, using both linear [33], and tree-structured [34] TAG representations.

4.4 Other extensions of GP using grammars

The typical grammar-based GP model is readily extended to test new GP ideas. For example, Wong and Leung [107] showed how logic grammars could readily emulate Automatically Defined Functions. McKay [54] used Ross' similar system to investigate partial function evaluation in GP. But Chomsky grammars are not the only source of such progress. Nguyen et al. [60] used TAGs to investigate biologically inspired operators in GP. McKay et al. [56] extended this to investigate the interaction between evolution, development and lifetime evaluation, applying evolutionary developmental systems in a form of layered learning.

In all these cases—and many others—the work could have used another GP formalism. The grammar simply made it easy to encode the extension, where other forms of GP would have required major re-coding to implement the same extension.

5 Alternative population-based stochastic search algorithms

GGGP is, in essence, a class of population-based stochastic search strategies (evolutionary algorithms) applied to a class of representations (grammar-based). Just as we may try different representations, as described in Sects. 2 to 4, so we may also try a range of search strategies. We outline some of these approaches.

5.1 Probabilistic model-building algorithms

Of the various alternative search strategies, by far the largest research effort to date has focused on algorithms that build a probabilistic model of the search space, then sample from that model to generate individuals to test for fitness. The fittest models are then used to update the probabilistic model. Algorithms falling into this class are known under a wide range of names, the most popular being "Estimation of Distribution Algorithms" (EDA) and "Ant Colony Algorithms" (ACO).[3] For detailed background in this area, and a survey of publications up to 2005, we refer the interested reader to Shan et al. [89]. For the sake of convenience, we refer to the general field as EDA-GP.

Whigham [103] built what would today be seen as a hybrid GP/EDA-GP system. It used a stochastic CFG, in which the success of individuals gradually biased the probabilities of productions, although the search still used mutation and crossover operators as well. At the time, PBIL [2], the first EDA had only just been published,

[3] While there may be substantial differences between EDA and ACO in general, in their application to GP search spaces, the differences have been largely a matter of terminology.

so the connection was not recognised. Tanev [95] used a similar strategy, based on stochastic CSGs, to handle change in a dynamic GP system.

Ratle and Sebag [78, 80] introduced SG-GP, the first pure grammar-based EDA-GP system, using CFGs; it was quickly followed by two others (both described as ant systems). Abbass et al. [1, 88] introduced Ant-TAG, which used tree adjoining grammars (TAG). Generalised Ant Programming (GAP) [42] used a stochastic CFG in a similar way.

All these systems used simple probability structures, in which only the probability tables are learnt, the grammar structure is not updated. This creates a dilemma. Too simple an initial grammar may make the problem impossible to solve; either because it does not represent important dependencies in the solution, so that it cannot converge to a state that generates the solution with high probability; or because it represents the wrong dependencies, so that the solution cannot be generated at all. On the other hand, a complex initial grammar leads to a tough parameter learning problem—the algorithm may converge too slowly to be useful.

Thus many of the more recent systems incorporate some form of grammar learning. Program Evolution with Explicit Learning (PEEL) [90] inferred both grammar structure and probabilities, though in a very limited way. Bosman and de Jong [4] introduced a system with similar structure-learning capabilities. Both use simple heuristics to update the grammar.

Grammar Model-based Program Evolution (GMPE) [91] takes a more systematic approach, using grammar learning methods from Natural Language Processing (NLP) to learn CFG structure. Hasegawa and Iba's PAGE (Programming with Annotated Grammar Estimation) [28] also uses NLP methods, but in this case, to learn grammars with latent annotations—a form of semantic grammar.

Research in these areas has been relatively successful, with authors demonstrating substantial improvements over evolutionary algorithms in terms of the number of fitness evaluations required to find a solution. However the computational complexity of the learning algorithms involved—especially in the more complex structure-learning algorithms—means that the running time of these algorithms are often long compared to a GP system. In the current state of the art, they are thus best-suited to problem domains with very expensive fitness evaluation, or in machine learning, to problems where the primary constraint is data availability rather than computation time.

5.2 Evolutionary developmental systems

EDS have seen an explosion of growth in recent years. In biology, Lindenmayer systems (a form of CFG) remain the almost universal mechanism for abstractly representing developmental processes, and have been widely used also in artificial EDS. In the field of complex adaptive systems, bracketed grammars (a restricted form of CFG) have been used to generate and study the evolution and dynamics of such systems with the analogy of chemical in the form of function [18].

In a different direction, McKay et al. [56] combined evolutionary/developmental search and layered learning in an algorithm (DTAG3P) that was successful on some difficult problems. However the integration of layered learning in this system means

that its performance is not directly comparable with that of GP and other similar systems. This may be viewed either as a disadvantage (DTAG3P requires more information than standard GP) or as an advantage (DTAG3P is able to use this additional structured information when it is available, whereas GP is not).

5.3 Other search strategies

The linear representation of Grammatical Evolution lends itself to ready adoption of search strategies from combinatorial optimisation. A wide range have been tried, with mixed success. O'Sullivan and Ryan [73] compared simulated annealing, hill climbing, and random and genetic search; genetic search performed substantially better overall, suggesting that its trade-off of exploration and exploitation is far better suited to GP search spaces than either the highly exploitative methods of simulated annealing and hill climbing, or the pure exploration of random search. O'Neill and Brabazon [63] didn't fare much better with differential evolution, though the performance of particle swarm algorithms [64] was closer to that of evolutionary algorithms. Taken together, though, these results suggest that it is not easy to improve on the performance of evolutionary search for the problem spaces that arise in grammar-based GP.

5.4 Meta-evolution

Hyperheuristics is now an important sub-area of evolutionary computing. It has received limited attention in GP, but three of the systems described here may be seen as learning hyperheuristics. Meta-GE uses a meta-grammar [69], which learns a grammar to describe solutions to an optimisation problem. GMPE [91] not only learns solutions to a problem, but also learns a grammar describing the solution space. Similarly, DTAG3P learns a grammar controlling the development of a series of solutions to problems of different sizes. All three systems exhibit an important characteristic: unlike most learning algorithms (which learn a solution to a specific problem), they learn a general method, allowing them to rapidly generate solutions to a new problem.

6 Applications

Grammatical approaches to GP have yielded a large and varied range of real-world applications. We present some highlights, in an eclectic mix from Ecological Modelling and Medicine to Finance and Design.

Evolutionary computation has demonstrated huge potential in design, creating solutions competing with, or even improving, those of human experts, and resulting in patentable inventions [50]. Coupling an evolutionary algorithm to a grammar representation is a particularly powerful departure for evolutionary design [36], with a wide range of applications including architecture [31], circuits [41, 52] and neural networks [24]. They feature a variety of grammars, including variants of Lindenmayer systems [31], Graph Grammars [17], and Shape Grammars [21, 71].

Wong and Leung [109] demonstrated the flexibility of their LOGENPRO engine on two medical datasets, mining production rules describing fracture admissions into a Hong Kong hospital, and diagnostic rules concerning scoliosis (curvature of the spine).

Computational finance presents significant real-world challenges to machine learning, arising from complexity, noise and constant change. Grammatical GP methods have proven particularly successful in financial modelling. For example, Brabazon and O'Neill have applied GE to develop rules for trading systems, and in more traditional classification problems such as credit rating and predicting corporate failure [5]. Tsakonas et al. [98] applied neural logic networks created by grammar-guided GP to bankruptcy prediction.

Ecological modelling uses spatial, temporal and spatio-temporal models, at a variety of scales. They are often expressed as partial differential or difference equations, symbolic rules, process models or finite-state machines. Early examples exploiting the language bias of GGGP modelled species distribution spatially [55, 105] and spatio-temporally [93]. Other work has modelled the time-series rainfall-runoff relationship for a number of catchments [106]. More recently, methods incorporating difference equations and the evolution of an equation component (such as the grazing term of a phytoplankton model) have been demonstrated [101]. These methods have been used to explore the quality of interpolated ecological time-series data [57].

Further illustrating the diversity of application, grammatical forms of GP have been applied in Bioinformatics (e.g., evolving regular expressions [49]), software cost estimation [92], robot control [96], music [9, 15], Logic Programming [44], fuzzy control [74] and solving differential equations [99].

7 The role of grammar-based GP

Grammars bring a number of benefits to GP. Undoubtedly the most important is a flexible means of search space restriction. Right from the early days of GP, Koza [47] emphasised the importance of closure: that every individual that could be created by the genetic operations must be capable of evaluation if expensive repair strategies are to be avoided. That is, the semantic restrictions of the search space must map onto syntactic restrictions. Of course, the simplest way to ensure this is to have no semantic restrictions: to guarantee that every feasible combinations of symbols can be interpreted. This was the original, and still a common, solution. But it is burdensome. It means that semantics must be provided for syntactic combinations such as $1 + \text{FALSE}$. This is generally manageable for relatively simple problems, but it can rapidly become unwieldy.

One option—commonly used in other areas of evolutionary computation—uses repair operators. Uninterpretable combinations are repaired so they can be evaluated. This works well for simple restrictions, but has rarely been attempted in GP, probably because of its complexity. One notable exception, partially grammar-based, is the linear GP of Keller and Banzhaf [46], which adopted a repair strategy to ensure the syntactic correctness of the programs generated by the system.

The alternative, providing a flexible mechanism to impose restrictions on the search space, was rapidly adopted—first, in the form of Strongly Typed Genetic Programming (STGP) [59], and subsequently and more flexibly, in a widening range of grammar representations. The genetic operators only need to be designed consistently with the syntactic requirements, so closure can be guaranteed without a need for repair. The complexity of possible restrictions depends on the form of grammar. For many problem domains, CFGs are sufficient. For example, CFGs are commonly used to define typing restrictions on languages—i.e. they implement STGP. However they are also used to represent more complex restrictions. One common use is through a form of incremental learning, where users gradually modify their grammars (typically via specialisation) as they increase their understanding of the problem domain.

We illustrate the value of this approach, encoding domain knowledge to reduce the search space to a feasible complexity, through an example, based on Ratle and Sebag's work on dimensionally-correct GP [79]. They use an idea from physics—that valid physical formulas must be dimensionally correct—to vastly prune the search space. They do so by imposing dimensional correctness through grammatical restrictions.[4]

Ratle and Sebag learnt complex physical formulas such as

$$u(t) = \frac{F}{K}\left(1 - e^{-\frac{Kt}{C_2}}\right) \tag{1}$$

$$u(t) = \frac{F}{K_1} + \frac{Ft}{C_1} + \frac{F}{K_2}\left(1 - e^{-\frac{Kt}{C_2}}\right). \tag{2}$$

However we will illustrate with an example based on the well-known Ideal Gas Law, which we hope will be familiar to our readers, in the form

$$P = \frac{RT}{V}. \tag{3}$$

We can imagine a situation (similar to that faced by the originators of these laws [20]), where a number of experiments have been conducted measuring the pressure (P), volume (V) and temperature (T) of a given amount of gas, and the data is available in tabular form. We might wish to find the form of an equation that would predict the pressure of the gas, given the temperature and volume—i.e. a formula such as Eq. 3. Today, we could just apply GP. In fact, almost any form of GP would work for Eq. 3, but for most there is little prospect of success with more complex formulae such as Eqs. 1 and 2. However we might recognise that evolving in the space of general expressions is unnecessary. Any physically realistic equation could be constructed as a sum of terms formed from multiplication, division and exponentiation. This could be readily expressed by a context-free grammar such as in Table 3.[1]

[4] While the ideas here are due to Ratle and Sebag, we use an equivalent representation closer to the logic grammars of Wong and Leung.

Table 3 Context free grammar for ideal gas law

Exp → Trm	Trm → Trm Mul Trm	Add → ''+''|'' − ''
Exp → Trm Add Trm	Trm → Var	Mul → ''×''|''/''
	Trm → Const	Var → ''T''|''V''

Of course, in this case we also need a mechanism to deal with the generation of constants from the Const nonterminal: any mechanisms that might be used in a standard GP may also be used here.

Again, this grammar readily finds the Ideal Gas Law; but as before, this restriction is inadequate to solve Ratle and Sebag's examples. But one may apply more semantic knowledge about physics: that valid formulae must be dimensionally correct. Since P—i.e. force per unit area—has dimensionality $mst^{-2} \times s^{-2} = ms^{-1}t^{-2}$, we may infer that the right hand side has the same dimensionality. V—the volume—has dimensionality s^3, and T—the temperature—has dimensionality ms^2t^{-2}. This imposes a very strong semantic restriction. It can't be converted in a general way to an equivalent context-free restriction, but it is readily represented in a context sensitive language such as the logic grammar shown in Table 4.[1] We represent the dimensionality of an expression by a vector $\mathbf{d} = (m, s, t)$, where each of m, s, t is the dimensionality of the corresponding physical quantity; $\oplus$ represents vector addition. This grammar would be called with the nonterminal Exp(1, −1, −2), representing an expression restricted to dimensionality ms^2t^{-2}. With restrictions such as these,[5] Ratle and Sebag were able to solve problems such as equations 1 and 2—problems which are far into the infeasible region for knowledge-free applications of GP.

It is worth pointing out the simplicity and generality of this solution, requiring only nine lines of very simple grammar productions. Note that, once coded, this grammar may be used to impose a similar dimensional constraint on any other physical problem; only the nonterminal with which it is called—here exp(1, −1, −2)—needs to be changed for a different problem. To impose a similar restriction in a non-grammar GP system would require many pages of complex code—if it could be written at all.

The most similar work we are aware of, using a non-grammar GP, is the series of papers of Rodriguez-Vazquez and Fleming, culminating in [82], where they evolve a model from a class known as "rational models", i.e. of the form shown in Eq. 4

$$y(k) = \frac{\sum_{i=1}^{\text{num}} P_{n_i}(k)\theta_{n_i}}{\sum_{j=1}^{\text{den}} P_{d_j}(k)\theta_{d_j}} \tag{4}$$

where each P_m is itself a polynomial of degree m in the arguments. This constraint, while much less complex than the dimensional constraint of Ratle and Sebag, must have required substantial coding, and at the time was seen as an important part of

[5] Actually, Ratle and Sebag also impose reasonable bounds on the allowable dimensionality of subexpressions—also readily expressible in logic grammars—but these are distractions to our illustrative purposes here.

Table 4 Logic grammar for ideal gas law

$\mathrm{Exp}(\mathbf{d}) \rightarrow \mathrm{Trm}(\mathbf{d})$	$\mathrm{Trm}(\mathbf{d_1} \oplus \mathbf{d_2}) \rightarrow \mathrm{Trm}(\mathbf{d_1})\ \mathrm{Mul}\ \mathrm{Trm}(\mathbf{d_2})$
$\mathrm{Exp}(\mathbf{d}) \rightarrow \mathrm{Trm}(\mathbf{d})\ \mathrm{Add}\ \mathrm{Trm}(\mathbf{d})$	$\mathrm{Trm}(\mathbf{d}) \rightarrow \mathrm{Var}(\mathbf{d})$
	$\mathrm{Trm}(0) \rightarrow \mathrm{Const}$
$\mathrm{Add} \rightarrow$ ''+''|'' − ''	$\mathrm{Mul} \rightarrow$ ''×''|''/''
$\mathrm{Var}(1, 2, -2) \rightarrow$ ''T''	$\mathrm{Var}(0, 2, 0) \rightarrow$ ''V''

the contribution of these papers. An equivalent grammar requires only a few lines of productions, as seen in Table 2.

It is important to note the user perspective here. The only additional complexity in using GGGP for these problems is the requirement to provide the grammar. The semantic interpretation of the symbols is provided in the same way as in a standard GP system (in the case of Ross' DCTG-GP, it is substantially simpler than for a standard GP system).

8 Characteristics, assumptions and limitations of grammars in evolutionary computation

The important place of grammars in modern Evolutionary Computation suggests that grammars bring some valuable advantages; however there are also associated costs. Here, we delineate these benefits and costs, and the assumptions that underly the marriage of grammars and evolutionary computation methods.

8.1 Characteristics of grammar systems in evolutionary computation

We consider, first, the use of grammars as representation languages for GP in general, looking at their general characteristics; then we look more specifically at the characteristics of specific grammar representations.

8.1.1 General characteristics: grammar as GP representation

Benefits

1. **Declarative search space restriction** Perhaps the most obvious consequence of using grammars in GP is the ability it provides to restrict the search space. This has been the primary justification for the use of GGGP almost from the start. The main benefit of restricting the search space is to reduce the search cost to the minimum necessary to find a solution, but it comes with the concomitant risk that the solution may not be within the search space defined by the grammar, or perhaps more insidious, that the solution may be isolated by the grammar constraints and may be difficult to reach.
   One common use has been to impose type restrictions; when used in this way, it is equivalent to Strongly Typed Genetic Programming (STGP) [59]. Another is to exclude solutions of a particular form—for some problems, solutions of

particular forms may have high fitness but low usefulness, so the simplest way to ensure that they do not confuse the search is to exclude them from the search space entirely.

In most GGGP systems, the grammar is one of the inputs to the system—provided in Backus-Naur Form or a similar formalism. Thus the search space is readily altered, simply by changing the grammar used. A common mode of operation with GGGP systems is to start with a very general grammar, and then to iteratively refine the grammar to narrow the search space as the results of earlier searches are obtained. This interactive style of use permits the user to influence the search process to a greater degree than is possible with many other forms of GP.

While narrowing of the search space—i.e. controlling the language bias—is the commonest mode of operation for these purposes, it is worth noting that the same search space (language) may be defined by a number of different grammars. Hence it is also possible to change the search space structure—its connectivity and overall fitness landscape—without changing the space itself. Thus it is possible to use change of grammar to alter the search bias, not only the language bias.

2. **Problem structure** A number of GP problem domains are themselves directly grammar-related. For example, protein motifs are essentially regular expressions delineating (with some level of reliability) a family of proteins. It is not surprising that grammar-based GP systems have figured prominently in motif discovery research.
3. **Homologous operators** To the extent that a grammar reflects the semantic structure of the domain, the homologous operators provided by GGGP replace one component of a solution with another having similar function. This homology reduces the likelihood of generating semantically meaningless code segments, and hence improves the search.
4. **Flexible extension** Many proposed extensions to GP can be readily implemented as grammar constraints. Practically, this makes GGGP systems powerful research tools for studying GP mechanisms, since proposed innovations can often be simply incorporated in a grammar and tested, where adding the same innovation to a classical GP system might require extensive recoding.

Costs These benefits of grammar-based approaches carry with them some costs, which we discuss below.

1. **Feasibility constraints** As with standard GP, grammar guided genetic programming is still far from resembling GA in its representational flexibility, despite some significant efforts in this direction. In tree-based GGGP systems using Chomsky grammars, it is even more difficult than in GP to design new operators, especially those making small local changes to the genotype. In these systems, any new operator has to meet not only the general requirements of tree consistency, but also any additional constraints imposed by the grammar. Thus subtree mutation and subtree crossover remain largely unchallenged in these areas.

   Some more recent representations alleviate this problem. TAG-based GP systems are an exception in this area (because of reduced constraint on allowable

operators), and a wide range of new operators have been defined for them. This is also the case for linear GGGP systems (including, we believe, the recent extensions of GE)—it is easy to design new operators, including local operators, in the genotype space.

2. **Repair mechanisms** When the genetic operators generate infeasible individuals, there are a number of possible options. Simply discarding the infeasible solutions is an acceptable approach if the level of infeasibility is low, or the cost of generating new individuals is small compared to the evaluation cost. However this isn't always the case in GP. A better alternative is repair, the method adopted in some versions of GE. It carries with it the potential risk of reducing the homology of crossover and mutation.
   In Chomsky-grammar tree-based systems, repair may be very difficult, and we are not aware of any systems using it. Repair is generally avoided by guaranteeing feasibility by severely restricting the genetic operators.
   TAG grammar systems avoid the problem in another way: feasibility is relatively easy to guarantee without much restriction on the GP operators, so repair is unnecessary.
3. **Limited flexibility** While GGGP systems are very useful for exploring new ideas when they can be incorporated into a grammar, the situation may sometimes be reversed when the extension is too complex to be encoded in this way. Then, it may be necessary to build the extension, not only into the underlying system, but also into any grammar that uses the extension.
4. **Turing incompleteness** The Turing completeness of a GGGP system depends upon the semantics of the grammar used, so that GGGP may deal with both Turing-complete and Turing-incomplete problems. Because of their general-purpose orientation, GGGP systems typically do not intrinsically offer any additional support for specific computational paradigms (recursion, iteration etc) such as is provided in some other GP systems. In a sense, this is just an instance of the preceding point.

8.1.2 Characteristics of specific grammar representations

1. **Incorporating semantics** A number of GGGP systems—Ross' DCTG-based system [83] and attribute-grammar systems [38, 110]—incorporate semantics along with the syntax of the grammar. As a result, these systems are highly retargetable. Applying them to a new problem requires only specification of the new grammar and semantics, and definition of the fitness function. Since the semantics may incorporate much of the complexity of the fitness function, and the latter is the only component requiring programming, the programming cost of targetting a new problem is generally small—in the case of DCTG-GP, quite typically ten or twenty lines of code. In incremental learning, where the grammar and semantics may be altered by the user to guide search, it is often the case that no re-programming is required at all.
2. **Operators** As discussed above, defining new operators—including local search operators and biologically motivated operators—is extremely difficult in

Chomsky-grammar tree GGGP systems because of the constrained search space, so most tree-based GGGP systems rely solely on subtree mutation and crossover. The linearization transformation of GE and like systems makes it easy to define such new operators, and to control their effect in the genotype space, but the relatively uncontrolled re-scaling resulting from the linearization transformation means that the effects in the phenotype space may be very different. The importance of this effect varies from grammar to grammar.

One of the key advantages claimed for TAG-based grammar systems [34] is the ease with which new operators may be defined. This, combined with the lack of disruption of the genotype-phenotype transformation, means that many of the advantages of the GA representation can be recaptured for GP.

3. **Long-distance dependencies** One of the key benefits of grammar-based GP is the homology of the operators, which can be viewed as providing less disruptive, and more meaningful, transformation of building blocks. In standard GGGP, the building blocks are connected sub-graphs of the derivation tree. While connectedness is clearly an important aspect of building blocks, it is arguable that many of the important structures in human-generated programs are not locally connected, and require long-distance dependencies just like those in natural language. The TAG transformation permits local dependencies in the genotype space (ie the TAG derivation tree) to map to long-distance dependencies in the intermediate phenotype space (CFG derivation trees) in a controlled way, corresponding to the structure of the grammar. For example, in TAG representation, the number dependence between "cat" and "sits" in "The cat which has just had a very filling lunch sits on the mat" is a local dependence, whereas it is long-distance in the corresponding CFG representation.
4. **Solution models** Grammars can define not only the very general search spaces required to describe the problem domain, but also more restricted spaces, even down to a single solution. Hence grammars, in common with some other representations, provide a mechanism to delineate the gradually narrowing set of potential solutions. However grammars were specifically designed as a way to represent constrained contextual relationships within families of trees, so it is not surprising that they have shown strengths in this area, providing the ability to incrementally learn the structure of problem solution spaces.
5. **Opportunities for search space transformation** Since the publication of the messy GA [23], search space transformation has been a core area of GA research. Outside grammar-based GP, it appears to have been less studied in the GP literature, perhaps because the constraints imposed by GP tree structure limit the available transformations. On the other hand, a wide range of transformations have been proposed for grammar-based representations, transforming the grammar derivation trees to linear strings. They rely on the restrictions imposed by the grammar rules, by imposing a numbering system on the productions of the grammar. These transformations offer the advantages of a linear genotype—ability to apply the wide range of tools and methods developed for GAs, and reduced constraint allowing much more flexible genetic operators. They also appear to offer parsimony advantages, with less bloat than is observed in tree-based GP systems.

However these advantages may come at a cost. For one thing, the operators may no longer be homologous after the linearization transformations. An alteration early in the genotype may change the interpretation of the subsequent genotype, so that (depending on the grammar) the proportion of highly disruptive operators may be higher than in the underlying tree-based representation; this in turn alters the exploration/exploitation balance of the algorithm, increasing the exploratory behaviour, potentially at the expense of fine-grained search later in the process.

8.2 Assumptions and limitations

The assumptions about the search space underlying all evolutionary applications include:

- That there is sufficient correlation between fitness and semantics, so that non-random search is useful
- That there is sufficient correlation between distance in the genotype and phenotype (semantic) spaces that evolutionary search is able to take advantage of the first correlation
- That these relationships are nevertheless sufficiently uneven (that is, the fitness landscape is sufficiently rough) that deterministic search methods do not perform well

Of course, these assumptions carry over to grammar-related applications: grammar-based GP systems will only perform well if the languages to be learnt satisfy these requirements.

Grammar-based GP systems impose further assumptions. All known to us rely on expression-tree semantics (i.e. the same representation as standard GP) for fitness evaluation. So they embody an assumption that the mapping between grammar space and expression trees also generates a correlation between the distance metrics. Since this mapping depends not only on the representation, but also on the specific grammar, there is considerable room for variation: one method may perform better than another with a specific grammar, yet may perform worse for the same problem if the grammar is changed, even if the language defined by the grammar remains fixed.

A concern is sometimes expressed, that grammar-based GP is weaker than GP, in that it can only be used for problems that can be represented by a grammar. This is almost entirely a misconception: grammar-guided GP is a direct generalisation of expression-tree GP; as Whigham showed [104], for any expression tree domain, there is a grammar in which crossover and mutation directly map to those in the expression-tree representation, so that the course of evolution in each will be identical. Only minute differences remain: the grammar-based and expression tree initialisations may be slightly different (though it would not be difficult to build an exact replica of expression tree representation); and for fitness functions defined directly on the expression-tree shape (usually, GP test problems rather than real problems), it is slightly more complex to define the fitness function.

The ability to encode knowledge about the problem remains one of the strongest justifications for the use of grammar-based GP. Perhaps the clearest demonstration of this remains Ratle and Sebag's work described in Sect. 7, in which the grammar constraint reduces the search space size by many orders of magnitude, turning a clearly infeasible problem into a feasible one. It is worth emphasising that this is not merely a matter of computational complexity, but also of data complexity. The grammar constraint not only reduces the search space size, it also reduces the VC dimensionality of the corresponding solution. Without the constraints imposed by the grammar, even if a search algorithm such as standard GP were able to *find* a solution of the complexity found by Ratle and Sebag's system, it would not be justified in *accepting* it: the data size required to justify acceptance of a formula of such complexity would be simply astronomical, without the grammar constraints.

Thus the representational strength of the language is important: most GGGP systems can accept at least context-free grammar languages; this is enough to encode a great deal of meta-knowledge about the solution space. Nevertheless, there remains an important place for representation languages (such as TAG and logic grammars) which can extend into the context-sensitive domain, and others (such as DCTG and attribute grammars) which can encode semantic knowledge. There is, as yet, no general consensus on the appropriate language power for real-world problems.

However this ability to reduce the search space through syntactic (and/or semantic) constraints comes with some concomitant risk. There is the obvious risk that the constraints may render the solution inaccessible: if the grammar is wrongly chosen, the solution may not even lie within the search space defined by the grammar. But there are more subtle risks as well. The constrained nature of the search space can render search more difficult. The grammar space is generally sparser than the corresponding expression-tree search space. Thus neighbourhoods may be sparser, and they may be less connected (that is, not only may there be fewer nodes in the search graph, but there may be proportionately even fewer links). Thus reachability of the solution may be an issue. This has already been shown by Daida [12] to be an important issue in expression-tree GP. While it has been little explored in grammar-based GP, it seems clear that the same issue can arise, but to an even greater degree, in grammar-based GP systems.

Two approaches perhaps alleviate this: linear and TAG-based grammar methods can claim some greater degree of connectivity. There has been some investigation [34, 84], but further study of these representational issues is clearly desirable.

9 Future directions

There is an expectation of review papers, that they will point the way to the future and outline what research directions will be important. These predictions are almost guaranteed to be wrong. The most important contributions of the future will be the innovative ideas that no-one foresees today. Thus cautioned, we here outline our vision of the medium-term future of grammar-based GP—say five to ten years.

Of course, we anticipate that grammar-based GP will share in much of the general future of the GP field. Progress in general understanding of GP behaviour,

and advances in GP technology, will generally translate directly into grammar-based GP (just as they have in the past). Since others are addressing these issues elsewhere in this volume, we concentrate on those aspects which are specific to grammar-based GP. In our view, they fall into three main areas: problem representation, algorithms and applications.

9.1 Grammars and problem representation

Research on different kinds of grammar will undoubtedly be important. We have already seen the basic CFG model extended to various CSGs; to stochastic grammars; and to a variety of semantic grammars. We don't expect this process to stop, and we do anticipate that some of these new grammars will have important properties for GP. Having chosen a grammar, one may represent its parse trees in various ways, either directly, or in a linearised form as in GE. We expect that alternative transformations for CFGs will continue to be explored, while transformations and linearisations for CSGs and semantic grammars are wide open for research.

Incorporating problem semantics into search is currently an active area of research in GP—both in representation of prior semantic knowledge, and incorporating that representation into the definition of the problem space, and in semantically-aware operators that either incorporate prior knowledge or that adapt to the semantics of the problem space as it is explored. In all of these areas, grammar representations offer the opportunity to directly represent the semantics, and to tie it to the syntax of the problem domain. We expect grammar-based GP to figure solidly in work in these areas.

There has been limited work so far on the fitness landscapes of various GGGP representations. We expect work in this area to expand substantially. In GGGP, it is a relatively trivial matter to rapidly change the problem representation; while there is a lot of practical experience in choosing grammars that work well, it would be desirable to have a much better theoretical understanding of the fitness-landscape effects of changing grammar representation.

Grammar representations entail a complex genotype-phenotype mapping; its continuity (causality) is important to the effectiveness of GP—a discontinuous mapping may represent a smooth phenotype landscape by a rough genotype landscape. There is plenty of room for increased understanding in this area to supplement the limited existing work, and to inform our choice of grammar representations and operators.

GGGP users typically experiment with grammar representations to find the most appropriate one for the problem domain. While experienced practitioners of each representation form have some tacit understanding of how to choose grammars, there is little explicit knowledge. More explicit knowledge may lead to more structured methodologies (and interactive software support) to incrementally find good representations for new problem domains, and even to partial or complete automation of the process.

GGGP has seen only limited application to Turing-complete problem domains. In some ways, this is surprising. A key difficulty in these domains is the need to evolve

two interlinked structures in parallel—in recursion, the epistasis between recursion and termination steps; in iteration, epistasis between increment and termination condition. Such dependencies are also common in human languages (for example, number- or gender-agreement in English), so many context-sensitive grammars provide mechanisms to support them, representing the common aspect by a single value. But this potential has seen only limited exploration so far.

9.2 Alternative population-based stochastic algorithms

The wider field of evolutionary algorithms has seen a huge increase in alternatives to the classical evolutionary algorithms in the past decade. This ferment is gradually spilling over into GP, with a range of non-classical algorithms being proposed. One key reason is the same as for GA: a desire to handle local and non-local dependencies between different components of the genome. Grammars offer substantial advantages. Non-local dependencies in an expression tree representation may become local in a suitably chosen grammar. Thus we expect GGGP to be relatively more important in application of non-classical algorithms than in the overall field of GP. We have already seen this in EDA-GP, where grammar models now form the bulk of new work. We anticipate a substantial flowering of work on the application of other search strategies to grammar models.

9.2.1 Evolutionary developmental systems

We anticipate that grammar-based systems will increase in importance in EDS. This is particularly likely to manifest itself in evolution of semantic grammars to control development. A key issue in biological development, largely unrealised in current EDS, is the interaction of the developmental process with the environment. This is important both for exogenous influence on development (e.g. the sex-determination of crocodile embryos by temperature) and for feedback mechanisms outside the control of genetics (e.g. the feedback-tuned co-development of lens and eye shape in higher animals). We believe semantic grammars have a role to play in this. A second potential area of application lies in the area of artificial chemistry, in extending the work of [18] to make use of more complex grammars, in particular stochastic grammars.

9.3 Applications

9.3.1 Real-world applications

While GGGP methods have been widely applied by practitioners and experts in the field, they are less known in the wider domains of engineering and science. There is thus a need to make these methods better known outside the relatively small GGGP community. One important factor is the required level of user knowledge. GAs have become widely known because they are so easy to implement, and require little knowledge to use reasonably competently. GP has had more limited exposure, perhaps because of the general need to program a fitness function—but some

applications, such as symbolic regression, can be used off-the-shelf, and in any case programming to this level is becoming a fairly widely-distributed skill, so that GP applications by non-experts are becoming more common. GGGP use, on the other hand, requires some familiarity with writing grammars. While this skill is widespread among computer scientists and linguists, it is not so common in other areas. Thus interactive mechanisms to support grammar development may be necessary if GGGP is to reach its full potential audience.

Grammars offer particular advantages for design problems, because they can directly represent many of the design constraints. We expect to see substantial work in this area, both in learning solutions to problem spaces described by grammars (i.e. continuing past directions), and in learning the grammars themselves (i.e. learning to describe particular classes of solutions).

9.3.2 Dynamic environments

Dynamic optimisation is currently one of the hottest areas in evolutionary computation. But optimisation problems aren't the only problems that change. In many real-world applications of GP, the objective also changes over time. Unlike optimisation problems, however, GP problems may vary in both their parameterisation and in their structure. Representations that can directly represent—and therefore explicitly change—the problem structure may be useful here. So we anticipate a flowering of research on GGGP application to dynamic problems. We also expect an important focus will lie in methods to adapt the grammar as the problem changes.

10 Summary and conclusions

Grammars offer many advantages in GP, resulting in their wide use. The most obvious lies in delineating the search space clearly, and avoiding unproductive search in infeasible regions. Yet this may be only scratching the surface. Homologous crossover and mutation seem important in supporting productive search, while grammar-based methods are yielding fruit in understanding the relationship between genetic operators and representations and in the traversal of different classes of fitness landscape. It also seems clear that much future work is required to gain a better understanding of the different effects of different grammar representations of the same problem domain. For example, the many-to-one mapping provided by grammar genotype-phenotype mappings may contribute substantially to the success of GGGP systems, but at present this relationship is not well understood. Grammar-based methods also offer important advantages in research, because of the ease with which they may be tailored to explore specific hypotheses.

A wide variety of grammar representations have been applied, using a number of different search strategies, to a great array of problems. We have only scratched the surface in this paper. We expect this to increase in the future, particularly if more human-engineered systems, supporting the users in developing suitable problem grammars, become available.

The field is still developing rapidly, with many important directions currently being explored. Perhaps the key areas are in representation and in search. In representation, there is important current work on alternative linearisation transformations, and on alternative grammar representations. In alternative search algorithms (especially EDA and ant-based algorithms), grammar representations offer clear advantages, and as a result grammar-based systems form one of the main threads. We look forward to substantial further progress in these areas over the coming decade.

Acknowledgments The authors thank Kwong Sak Leung, Man Leung Wong and Brian Ross for insightful discussions that helped to form their perspectives on grammar-based GP, Kee Siong Ng for his suggestions at the final stage of editing. Thanks are also due to the anonymous referees, who helped us to shape the discussion more comprehensibly. Seoul National University Institute for Computer Technology provided some of the research facilities for this study, which was also supported by a Korea Research Foundation Grant funded by the Korean Government (KRF-2008-313-D00943). MO'N thanks Science Foundation Ireland for support under Grant No. 08\IN.1\I1868. NXH was partly funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01.14.09 for this work.

References

1. H.A. Abbass, N.X. Hoai, R.I. McKay, AntTAG: a new method to compose computer programs using colonies of ants. in *The IEEE Congress on Evolutionary Computation* (2002), pp. 1654–1659
2. S. Baluja, *Population-Based Incremental Learning: A Method For Integrating Genetic Search Based Function Optimization and Competitive Learning*. Tech. Rep. CMU-CS-94-163, (Carnegie Mellon University, 1994)
3. W. Bohm, A. Geyer-Schulz, Exact uniform initialization for genetic programming. in *Foundations of Genetic Algorithms IV*, ed. by R.K. Belew, M. Vose (Morgan Kaufmann, University of San Diego, CA, USA, 1996), pp. 379–407
4. P.A.N. Bosman, E.D. de Jong, Grammar transformations in an EDA for genetic programming, in *Special Session: OBUPM—Optimization by Building and Using Probabilistic Models* (GECCO., Seattle, Washington, USA, 2004)
5. A. Brabazon, M. O'Neill, *Biologically Inspired Algorithms for Financial Modelling. Natural Computing Series* (Springer, Berlin, 2006)
6. P. Bruhn, A. Geyer-Schulz, Genetic programming over context-free languages with linear constraints for the knapsack problem: first results. Evol. Comput. **10**(1), 51–74 (2002)
7. H. Christiansen, A survey of adaptable grammars. SIGPLAN Not. **25**(11), 35–44 (1990)
8. R. Cleary, M. O'Neill, An attribute grammar decoder for the 01 multiconstrained knapsack problem. in *Evolutionary Computation in Combinatorial Optimization—EvoCOP 2005*, LNCS, vol. 3448, ed. by G.R. Raidl, J. Gottlieb (Springer, Lausanne, Switzerland, 2005), pp. 34–45
9. D. Costelloe, C. Ryan, Towards models of user preferences in interactive musical evolution. in *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, vol. 2, ed. by D. Thierens, H.G. Beyer, J. Bongard, J. Branke, J.A. Clark, D. Cliff, C.B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J.F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K.O. Stanley, T. Stutzle, R.A. Watson, I. Wegener (ACM Press, London, 2007), pp. 2254–2254
10. N.L. Cramer, A representation for the adaptive generation of simple sequential programs. in *Proceedings of an International Conference on Genetic Algorithms and the Applications*, ed. by J.J. Grefenstette (Carnegie Mellon University, Pittsburgh, PA, 1985), pp. 183–187
11. M. de la Cruz Echeandía, A.O. de la Puente, M. Alfonseca, Attribute grammar evolution. in *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, Lecture Notes in Computer Science, vol. 3562 (Springer, Berlin, 2005), pp. 182–191
12. J.M. Daida, H. Li, R. Tang, A.M. Hilss, What makes a problem GP-hard? validating a hypothesis of structural causes. in *Genetic and Evolutionary Computation—GECCO-2003*, LNCS, vol. 2724, 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 (Springer, Chicago, 2003), pp. 1665–1677
13. A.L. abu Dalhoum, M. al Zoubi, M. de la Cruz, A. Ortega, M. Alfonseca, A genetic algorithm for solving the p-median problem. in *2005 European Simulation and Modeling Conference (ESM2005)* (Oporto, 2005)
14. R. Dawkins, *The Blind Watchmaker* (Penguin, Harmondsworth, 1991)
15. A.O. de la Puente, R.S. Alfonso, M.A. Moreno, Automatic composition of music by means of grammatical evolution. in *Proceedings of the 2002 Conference on APL* (ACM Press, Madrid, Spain, 2002), pp. 148–155
16. I. Dempsey, M. O'Neill, A. Brabazon, *Foundations in Grammatical Evolution for Dynamic Environments* (Springer, Berlin, 2009)
17. H. Ehrig, M. Pfender, H. Schneider, Graph-grammars: an algebraic approach. in *Proceedings of IEEE Conference on Automata and Switching Theory* (1973), pp. 167–180
18. W. Fontana, Algorithmic chemistry. in *Artificial Life II*, vol. 2, ed. by C. Langton, C. Taylor, J. Farmer, S. Rasmussen (Addison-Wesley, Reading, 1991), pp. 159–209
19. J.J. Freeman, A linear representation for gp using context free grammars. in *Proceedings of Genetic Programming 1998* (Morgan Kaufmann, Los Altos, 1998), pp. 72–77
20. J.L. Gay-Lussac, Recherches sur la dilatation des gaz et des vapeurs. Ann. de Chimie **63**(137) (1802)
21. J. Gero, Evolutionary learning of novel grammars for design improvement. AIEDAM **8**(2), 83–94 (1994)
22. A. Geyer-Schulz, *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning, Studies in Fuzziness*, vol. 3, 2 edn. (Physica Verlag, Heidelberg, Germany, 1996)
23. D. Goldberg, B. Korb, K. Deb, Messy genetic algorithms: motivation, analysis and first results. Complex Syst. **3**, 493–530 (1989)
24. F. Gruau, Automatic definition of modular neural networks. Adapt. Behav. **3**(2), 151–183 (1994)
25. F. Gruau, On using syntactic constraints with genetic programming. in *Advances in Genetic Programming 2*, Chap. 19, ed. by P.J. Angeline, K.E. Kinnear Jr. (MIT Press, Cambridge, MA, 1996), pp. 377–394
26. R. Harper, A. Blair, A structure preserving crossover in grammatical evolution. in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 3, 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 (IEEE Press, Edinburgh, UK, 2005), pp. 2537–2544
27. R. Harper, A. Blair, Dynamically defined functions in grammatical evolution. in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation* (IEEE Press, Vancouver, 2006), pp. 2638–2645
28. Y. Hasegawa, H. Iba, *Estimation of distribution algorithm based on probabilistic grammar with latent annotations. Evolutionary Computation, 2007. CEC 2007*. IEEE Congress on (25–28 Sept. 2007), pp. 1043–1050
29. E. Hemberg, C. Gilligan, M. O'Neill, A. Brabazon, A grammatical genetic programming approach to modularity in genetic programming. in *Proceedings of the Tenth European Conference on Genetic Programming 2007, vol. LNCS* (Springer, Valencia, 2007)
30. M. Hemberg, U.M. O'Reilly, Extending grammatical evolution to evolve digital surfaces with genr8. in *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, LNCS, vol. 3003, ed. by M. Keijzer, U.M. O'Reilly, S.M. Lucas, E. Costa, T. Soule (Springer, Coimbra, 2004), pp. 299–308
31. M. Hemberg, U.M. O'Reilly, A. Menges, K. Jonas, M. da Costa Gonçalves, S.R. Fuchs, Genr8: architects' experience with an emergent design tool. in *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, ed. by J. Romero, P. Machado (Springer, Berlin, Heidelberg, 2007), pp. 167–188
32. J. Hicklin, *Application of the genetic algorithm to automatic program generation*. Master's Thesis, (University of Idaho, Moscow, ID, 1986)
33. N.X. Hoai, Solving the symbolic regression with tree-adjunct grammar guided genetic programming: the preliminary results. in *Australasia-Japan Workshop on Intelligent and Evolutionary Systems*, ed. by N. Kasabov, P. Whigham (University of Otago, Dunedin, 2001)
34. N.X. Hoai, R.I. McKay, D. Essam, Representation and structural difficulty in genetic programming. IEEE Trans. Evol. Comput. **10**(2), 157–166 (2006)

35. P. Holmes, P.J. Barclay, Functional languages on linear chromosomes, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, ed. by J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (MIT Press, Stanford University, CA, USA, 1996), p. 427
36. G.S. Hornby, Functional scalability through generative representations: the evolution of table designs. Environ. Plann. B: Plann. Des. **31**(4), 569–587 (2004)
37. T.S. Hussain, R.A. Browse, Attribute grammars for genetic representations of neural networks and syntactic constraints of genetic programming. in *Workshop on Evolutionary Computation. Held at the 12 Canadian Conference on Artificial Intelligence* (Vancouver, Canada, 1998)
38. T.S. Hussain, R.A. Browse, Basic properties of attribute grammar encoding. in *Late Breaking Papers at the Genetic Programming 1998 Conference*, ed. by J.R. Koza (Stanford University Bookstore, University of Wisconsin, Madison, Wisconsin, USA, 1998)
39. C.M. Johnson, S. Feyock, A genetics-based technique for the automated acquisition of expert system rule bases. in *Proceedings of the IEEE/ACM International Conference on Developing and Managing Expert System Programs* (IEEE Computer Society Press, Los Alamitos, 1991), pp. 78–82.
40. A. Joshi, L. Levy, M. Takahashi, Tree adjunct grammars. J. Comput. Syst. Sci. **10**, 136–163 (1975)
41. U.R. Karpuzcu, Automatic verilog code generation through grammatical evolution. in *Genetic and Evolutionary Computation Conference (GECCO2005) Workshop Program*, ed. by F. Rothlauf, M. Blowers, J. Branke, S. Cagnoni, I.I. Garibay, O. Garibay, J. Grahl, G. Hornby, E.D. de Jong, T. Kovacs, S. Kumar, C.F. Lima, X. Llorà, F. Lobo, L.D. Merkle, J. Miller, J.H. Moore, M. O'Neill, M. Pelikan, T.P. Riopka, M.D. Ritchie, K. Sastry, S.L. Smith, H. Stringer, K. Takadama, M. Toussaint, S.C. Upton, A.H. Wright (ACM Press, Washington, DC, 2005), pp. 394–397
42. C. Keber, M.G. Schuster, Option valuation with generalized ant programming. in *Proceedings of the Genetic and Evolutionary Computation Conference* (Morgan Kaufmann Publishers Inc., Los Altos, 2002), pp. 74–81
43. M. Keijzer, V. Babovic, Dimensionally aware genetic programming. in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2, ed. by W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith (Morgan Kaufmann, Orlando, Florida, 1999), pp. 1069–1076
44. Keijzer, M., Babovic, V., Ryan, C., O'Neill, M., Cattolico, M.: Adaptive logic programming. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-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, San Francisco, California, 2001), pp. 42–49
45. 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*, LNCS, vol. 2278, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi (Springer, Kinsale, Ireland, 2002), pp. 123–130
46. R.E. Keller, W. Banzhaf, Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. in *Genetic Programming 1996: Proceedings of the First Annual Conference*, ed. by J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (MIT Press, Stanford University, CA, USA, 1996), pp. 116–122
47. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992)
48. J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press, Cambridge, Massachusetts, 1994)
49. J.R. Koza, D. Andre, F.H. Bennett III, M. Keane, *Genetic Programming 3: Darwinian Invention and Problem Solving* (Morgan Kaufmann, Los Altos, 1999)
50. 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 Publishers, Dordrecht, 2003)
51. R.M. MacCallum, Introducing a Perl genetic programming system: and can meta-evolution solve the bloat problem? in *Genetic Programming, Proceedings of EuroGP'2003*, LNCS, vol. 2610, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa (Springer, Essex, 2003), pp. 364–373
52. T. McConaghy, G. Gielen, Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions. in *GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, vol. 1, ed. by M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M.V. Butz, C. Coello Coello, D. Dasgupta, S.G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, F. Rothlauf, C. Ryan, D. Thierens (ACM Press, Seattle, 2006), pp. 855–862

53. R. McGee, M. O'Neill, A. Brabazon, The syntax of stock selection: grammatical evolution of a stock picking model. in *IEEE World Congress on Computational Intelligence WCCI 2010* (IEEE Press, Barcelona, 2010)
54. B. McKay, Partial functions in fitness-shared genetic programming. in *Proceedings of the 2000 Congress on Evolutionary Computation CEC00* (IEEE Press, La Jolla Marriott Hotel La Jolla, California, 2000), pp. 349–356
55. R.I. McKay, Variants of genetic programming for species distribution modelling—fitness sharing, partial functions, population evaluation. Ecol. Model. **146**(1–3), 231–241 (2001)
56. R.I. McKay, T.H. Hoang, D.L. Essam, X.H. Nguyen, Developmental evaluation in genetic programming: the preliminary results. in *Proceedings of the 9th European Conference on Genetic Programming*, Lecture Notes in Computer Science, vol. 3905, ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Springer, Budapest, Hungary, 2006), pp. 280–289
57. R.I. McKay, T.H. Hoang, N. Mori, X.H. Nguyen, D.L. Essam, Model-building with interpolated temporal data. Ecol. Inform. **1**(3), 259–268 (2006)
58. J.F. Miller, P. Thomson, Cartesian genetic programming. in *Proceedings of the European Conference on Genetic Programming* (Springer, London, 2000), pp. 121–132
59. D.J. Montana, Strongly typed genetic programming. Evol. Comput. **3**(2), 199–230 (1995)
60. X.H. Nguyen, R.I. McKay, D.L. Essam, H.A. Abbass, Genetic transposition in tree-adjoining grammar guided genetic programming: the relocation operator. in *2004 Asia-Pacific Conference on Simulated Evolution and Learning* (2004)
61. P. Nordin, W. Banzhaf, F.D. Francone, Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. in *Advances in Genetic Programming 3*, Chap. 12, ed. by L. Spector, W.B. Langdon, U.M. O'Reilly, P.J. Angeline (MIT Press, Cambridge, 1999), pp. 275–299
62. M. O'Neill, A. Brabazon, mGGA: the meta-grammar genetic algorithm. in *Proceedings of the 8th European Conference on Genetic Programming*, Lecture Notes in Computer Science, vol. 3447, ed. by M. Keijzer, A. Tettamanzi, P. Collet, J.I. van Hemert, M. Tomassini (Springer, Lausanne, 2005), pp. 311–320
63. M. O'Neill, A. Brabazon, Grammatical differential evolution. in *Proceedings of the International Conference on Artificial Intelligence* (CSEA Press, Las Vegas, 2006), pp. 231–236
64. M. O'Neill, A. Brabazon, Grammatical swarm: the generation of programs by social programming. Nat. Comput. **5**(4), 443–462 (2006)
65. M. O'Neill, A. Brabazon, M. Nicolau, S.M. Garraghy, P. Keenan, pi grammatical evolution. in *Genetic and Evolutionary Computation—GECCO-2004, Part II*, Lecture Notes in Computer Science, vol. 3103, ed. by K. Deb, R. Poli, W. Banzhaf, H.G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P.L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, A. Tyrrell (Springer, Seattle, 2004), pp. 617–629
66. M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, A. Brabazon, Geva: grammatical evolution in java. SIGEVOlution **3**(2), 17–23 (2008)
67. M. O'Neill, C. Ryan, Grammatical evolution. IEEE Trans. Evol. Comput. **5**(4), 349–358 (2001)
68. M. O'Neill, C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, Genetic Programming*, vol. 4. (Kluwer Academic Publishers, Dordrecht, 2003) URL http://www.wkap.nl/prod/b/1-4020-7444-1
69. 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*, LNCS, vol. 3003, ed. by M. Keijzer, U.M. O'Reilly, S.M. Lucas, E. Costa, T. Soule (Springer, Coimbra, 2004), pp. 138–149
70. M. O'Neill, C. Ryan, M. Keijzer, M. Cattolico, Crossover in grammatical evolution. Genet. Programm. Evol. Mach. **4**(1), 67–93 (2003)
71. M. O'Neill, J.M. Swafford, J. McDermott, J. Byrne, A. Brabazon, E. Shotton, C. McNally, M. Hemberg, Shape grammars and grammatical evolution for evolutionary design. in *GECCO 2009: Genetic and Evolutionary Computation Conference* (ACM, Montreal, 2009)
72. A. Ortega, M. de la Cruz, M. Alfonseca, Christiansen grammar evolution: grammatical evolution with semantics. IEEE Trans. Evol. Comput. **11**(1), 77–90 (2007)
73. 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*, LNCS, vol. 2278, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi (Springer, Kinsale, 2002), pp. 268–277

74. J. Otero-Rodriguez, S. Garcia-Carbajal, L. Sanchez-Ramos, Fuzzy control applied to a gas transport network in a siderurgycal environment. in *7th International Conference in Information Processing and Management of Uncertainty in Knowledge Based Systems* (Paris, 1998), pp. 403–410
75. N. Paterson, M. Livesey, Evolving caching algorithms in C by genetic programming. in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, ed. by J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo (Morgan Kaufmann, Stanford University, CA, USA, 1997), pp. 262–267
76. N.R. Paterson, M. Livesey, Distinguishing genotype and phenotype in genetic programming. in *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28–31, 1996*, ed. by J.R. Koza (Stanford Bookstore, Stanford University, CA, USA, 1996), pp. 141–150
77. A. Ratle, M. Sebag, Genetic programming and domain knowledge: beyond the limitations of grammar-guided machine discovery. in *Parallel Problem Solving from Nature—PPSN VI 6th International Conference*, LNCS, vol. 1917, ed. by M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, H.P. Schwefel (Springer, Paris, 2000), pp. 211–220
78. A. Ratle, M. Sebag, Avoiding the bloat with probabilistic grammar-guided genetic programming. in *Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001*, LNCS, vol. 2310, ed. by P. Collet, C. Fonlupt, J.K. Hao, E. Lutton, M. Schoenauer (Springer, Creusot, 2001), pp. 255–266
79. A. Ratle, M. Sebag, Grammar-guided genetic programming and dimensional consistency: application to non-parametric identification in mechanics. Appl. Soft Comput. **1**(1), 105–118 (2001)
80. A. Ratle, M. Sebag, A novel approach to machine discovery: Genetic programming and stochastic grammars. in *Proceedings of Twelfth International Conference on Inductive Logic Programming*, LNCS, vol. 2583, ed. by S. Matwin, C. Sammut (Springer, Sydney, Australia, 2003), pp. 207–222
81. M. Ridley, *Evolution* (Blackwell Science, London, 1996)
82. K. Rodríguez-Vázquez, P.J. Fleming, Use of genetic programming in the identification of rational model structures. in *Proceedings of the European Conference on Genetic Programming*, Lecture Notes in Computer Science, vol. 1802 (Springer, London, 2000), pp. 181–192
83. B.J. Ross, Logic-based genetic programming with definite clause translation grammars. New Generation Comput. **19**(4), 313–337 (2001)
84. F. Rothlauf, M. Oetzel, *On the locality of grammatical evolution*. Working Paper 11/2005, Department of Business Administration and Information Systems, University of Mannheim, D-68131 Mannheim, Germany (2005)
85. 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*, ed. by A.M. Barry (AAAI, Chigaco, 2003), pp. 142–145
86. C. Ryan, M. Nicolau, M. O'Neill, Genetic algorithms using grammatical evolution. in *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, LNCS, vol. 2278, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi (Springer, Kinsale, 2002), pp. 278–287
87. C. Ryan, M. O'Neill, A. Azad, No coercion and no prohibition—A position independent encoding scheme for evolutionary algorithms. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-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, San Francisco, 2001), p. 187
88. Y. Shan, H. Abbass, R.I. McKay, D. Essam, AntTAG: a further study. in *Proceedings of the Sixth Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, ed. by R. Sarker, B. McKay (Australian National University, Canberra, 2002)
89. Y. Shan, R. McKay, D. Essam, H. Abbass, A survey of probabilistic model building genetic programming. in *Scalable Optimization via Probabilistic Modeling, Studies in Computational Intelligence*, vol. 33, ed. by M. Pelikan, K. Sastry, E. Cantu-Paz (Springer, Berlin, 2006), pp. 121–160
90. Y. Shan, R.I. McKay, H.A. Abbass, D. Essam, Program evolution with explicit learning: a new framework for program automatic synthesis. in *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, ed. by R. Sarker, R. Reynolds, H. Abbass, K.C. Tan, B. McKay, D. Essam, T. Gedeon (IEEE Press, Canberra, 2003), pp. 1639–1646
91. Y. Shan, R.I. McKay, R. Baxter, H. Abbass, D. Essam, N.X. Hoai, Grammar model-based program evolution. in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation* (IEEE Press, Portland, Oregon, 2004), pp. 478–485

92. Y. Shan, R.I. McKay, C.J. Lokan, D.L. Essam, Software project effort estimation using genetic programming. in *Proceedings of International Conference on Communications Circuits and Systems* (2002)
93. Y. Shan, D. Paull, R.I. McKay, Machine learning of poorly predictable ecological data. Ecological Modelling **195**(1–2), 129–138 (2006). Selected Papers from the Third Conference of the International Society for Ecological Informatics (ISEI), August 26–30, 2002, Grottaferrata, Rome, Italy
94. L. Spector, K. Stoffel, Ontogenetic programming. in *Genetic Programming 1996: Proceedings of the First Annual Conference*, ed. by J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (MIT Press, Stanford University, CA, USA, 1996), pp. 394–399
95. I. Tanev, Implications of incorporating learning probabilistic context-sensitive grammar in genetic programming on evolvability of adaptive locomotion gaits of snakebot. in *Proceedings of GECCO 2004* (Seattle, Washington, 2004)
96. I. Tanev, Genetic programming incorporating biased mutation for evolution and adaptation of snakebot. Genet. Programm. Evol. Mach. **8**(1), 39–59 (2007)
97. I. Tanev, K. Shimohara, On role of implicit interaction and explicit communications in emergence of social behavior in continuous predators-prey pursuit problem. in *Genetic and Evolutionary Computation—GECCO-2003*, LNCS, vol. 2724, 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 (Springer, Berlin, 2003), pp. 74–85
98. A. Tsakonas, G. Dounias, M. Doumpos, C. Zopounidis, Bankruptcy prediction with neural logic networks by means of grammar-guided genetic programming. Expert Systems With Applications **30**(3), 449–461 (2006). Intelligent Information Systems for Financial Engineering
99. I.G. Tsoulos, I.E. Lagaris, Solving differential equations with genetic programming. Genet. Programm. Evol. Mach. **7**(1), 33–54 (2006)
100. R. Vanyi, S. Zvada, Avoiding syntactically incorrect individuals via parameterized operators applied on derivation trees. Congr. Evol. Comput. (CEC) **4**, 2791–2798 (2003)
101. P. Whigham, G. Dick, F. Recknagel, Exploring seasonal patterns using process modelling and evolutionary computation. Ecol. Model. **195**(1–2), 146–152 (2006)
102. P.A. Whigham, Grammatically-based genetic programming. in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, ed. by J.P. Rosca (Tahoe City, California, USA, 1995), pp. 33–41
103. P.A. Whigham, Inductive bias and genetic programming. in *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, vol. 414, ed. by A.M.S. Zalzala (IEE, Sheffield, 1995), pp. 461–466
104. 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 (1996) URL http://www.divcom.otago.ac.nz/sirc/Peterw/Publications/thesis.zip
105. P.A. Whigham, Induction of a marsupial density model using genetic programming and spatial relationships. Ecol. Model. **131**(2–3), 299–317 (2000)
106. P.A. Whigham, P.F. Crapper, Time series modelling using genetic programming: An application to rainfall-runoff models. in *Advances in Genetic Programming 3*, Chap. 5, ed. by L. Spector, W.B. Langdon, U.M. O'Reilly, P.J. Angeline (MIT Press, Cambridge, 1999), pp. 89–104
107. M.L. Wong, K.S. Leung, Applying logic grammars to induce sub-functions in genetic programming. in *1995 IEEE Conference on Evolutionary Computation*, vol. 2 (IEEE Press, Perth, 1995), pp. 737–740
108. M.L. Wong, K.S. Leung, Evolutionary program induction directed by logic grammars. Evol. Comput. **5**(2), 143–180 (1997)
109. M.L. Wong, K.S. Leung, *Data Mining Using Grammar Based Genetic Programming and Applications, Genetic Programming*, vol. 3 (Kluwer Academic Publishers, Dordrecht, 2000)
110. S. Zvada, R. Vanyi, Improving grammer based evolution algorithms via attributed derivation trees. in *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, LNCS, vol. 3003, ed. by M. Keijzer, U.M. O'Reilly, S.M. Lucas, E. Costa, T. Soule (Springer, Coimbra, 2004), pp. 208–219