Section 1  Grammars in Genetic Programming: A Brief Review

MCKAY, Robert I¹, NGUYEN, Xuan Hoai¹², WHIGHAM, Peter Alexander³ & SHAN, Yin¹
(¹ School of Information Technology and Electrical Engineering, University of New South Wales at the Australian Defence Force Academy, Canberra 2600, Australia)
(² Department of Computer Science, Army Technical Academy, Hanoi, Viet Nam)
(³ Department of Information Science, University of Otago, Dunedin, New Zealand)

Abstract: Grammars have become one of the most important formalisms in the field of genetic programming. We briefly survey the history of their application in genetic programming, and discuss their relevance to the field.

Key words: genetic programming, grammar, context free, tree adjunct, estimation of distributions

1  Introduction

Genetic Programming (GP) was first described by Cramer¹ and fully developed by Koza². Over the succeeding years, it has become increasingly important, and has been applied to a wide range of problems. In essence, GP uses evolutionary search methods to search an in-principle unbounded space of expressions for solutions to problems.

GP is important as one of the few modelling techniques able to learn solutions of potentially unbounded complexity, as opposed to the bounded complexity assumed by most methods. The other approaches able to naturally handle unbounded complexity approaches are Inductive Logic Programming (ILP) and Recurrent Neural Networks (RNNs) – in the ILP literature, the distinction is often drawn by contrasting first-order expression languages with zeroth order, or attribute-based, expression languages. Along with ILP, GP offers white-box, human interpretable solutions, as opposed to the black-box solutions provided by RNNs, so that ILP forms its most direct competitor in domains where human-comprehensible solutions are desired. While the relative strengths of ILP and GP have not been studied extensively (with the limited exception of Tang³), principally because their different representation languages render it difficult to guarantee fair comparisons, it appears that ILP has been most successfully applied in learning in relatively cleanly constrained domains, while GP has demonstrated strength primarily in noisy, complex, and poorly delineated search domains. GP has been successfully applied in a range of real-World problems, ranging from ecological modelling⁴ to software cost estimation⁵, from radar antenna design⁶ to quantum circuit design⁷, from DNA motif detection⁸ to digital logic design⁹.

Grammars have been used with GP almost from its inception; at first, grammar-based approaches formed a small segment of the GP world, but their role has expanded to the point where Grammatical Evolution (GE¹⁰) is now one of the most widely applied GP methods, and in particular areas, notably Estimation of Distribution Algorithms for GP, grammar-based approaches constitute the majority of the current research effort. However the literature on grammar-based GP is widely scattered, with significant portions only available in difficult-to-obtain conference proceedings, resulting in many researchers (including ourselves) re-discovering the wheel.

It seems that grammar-based GP has now reached a stage of maturity justifying – indeed, almost demanding – a concerted effort to draw together the threads of the field. In section 2, we will provide a historical framework for the field, and fit the published work into this framework. In section 3, we will attempt to unravel the advantages – and disadvantages – that grammar based approaches as a whole, and particular implementations in more detail, bring to GP. We will summarise our conclusions in section 4.
2 History

2.1 Foreshadowings

Antonisse\textsuperscript{10} appears to be the first researcher to propose the use of context free grammars for generating grammar-based chromosomes in GAs. The proposed system, known as grammar-based GAs, uses as chromosomes the strings from the language generated by a CFG $G$. When crossover is to be carried out between two chosen individuals, they must be parsed into two derivation trees of $G$. Then, the crossover acts in a similar way as in the Grammar-Guided GP (GGGP) systems, using a tree based representation, described in the next subsection.

The use of the string set as the chromosome, and the consequent need to parse strings, creates two important difficulties in this approach (perhaps explaining the dearth of implementations). Firstly, parsing is an expensive (and as will subsequently be seen, unnecessary) process, greatly increasing the computational cost of this approach. Secondly, if the grammar is ambiguous, for each string, there might be more than one derivation tree. Therefore, Antonisse suggested that the use of ambiguous grammars be forbidden. He argued that this restriction was not problematic, as ambiguous grammars would be bad for evolution in any case – a conclusion cast in considerable doubt by a recent publication of Nguyen et al\textsuperscript{11}.

Some years after Antonisse’ proposal, a number of researchers suggested that grammars might be used to control the structure of programs in GP. Stefanski\textsuperscript{12} proposed the use of abstract syntax trees to set a declarative bias for GP; Roston\textsuperscript{13} demonstrated how a formal grammar might be used to specify constraints for GP in the context of engineering design; and Mizoguchi et al\textsuperscript{14} suggested the use of production rules to generate hardware language descriptions during the evolutionary process. Roston used the grammar to generate the initial population only, while the other proposals gave only a theoretical description and do not appear to have been implemented. None of the systems maintained the derivation trees of the grammar, either re-parsing from the string set as for Antonisse (Stefanski) or combining with Strongly Typed GP\textsuperscript{15} (Roston).

2.2 Grammar-Guided Genetic Programming

2.2.1 Tree Based GGGP

The first fully-fledged GGGP systems were independently invented at about the same time by three different researchers. Whigham\textsuperscript{16,17} proposed the CFG-GP system, in which a context-free grammar was used to generate the population, consisting of derivation trees of the CFG. Schultz\textsuperscript{18} derived his very similar GGGP system for learning knowledge rules for expert systems, which differs mainly in the algorithm used to initialise the population\textsuperscript{19}. Wong and Leung\textsuperscript{20-24} 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. This is the primary difference between the systems; in other respects, LOGENPRO and CFG-GP are very similar. Given the similarity of the three systems, we base the description below primarily on CFG-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 $G$

2. **Procedure for initialising population.** Whigham\textsuperscript{17} 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 Schultz\textsuperscript{19} derived an algorithm for initialising the population based on the derivation-step-uniform distribution. The initialisation procedure for LOGENPRO was ‘embedded in PROLOG’\textsuperscript{27}

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.

4. **Genetic Operators** The genetic operators are the selection mechanism, reproduction, crossover, and
mutation. Selection and reproduction are as in standard GP.

• In crossover, two internal nodes labelled with the same nonterminal symbol of the grammar are chosen at random, and the two sub-derivation trees underneath them are 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. Genetic recombinations in biological systems are usually homologous – the genetic materials are not exchanged in completely random fashion, but between genes that have similar function. By analogy, we term crossovers exchanging sub-derivation trees stemming from the same nonterminal symbols ‘homologous crossovers’.

• Mutation is performed by selecting an internal node at random. The nonterminal on this node is noted, and the sub-derivation tree rooted there is deleted. A new sub-derivation tree starting from the same nonterminal, and generating according to the grammar, is randomly generated to replace the deleted one.

5. Parameters Parameters include population size (POPSIZE), maximal number of generations (MAXGEN), maximal depth for the individuals, and probabilities for the application of the operators. Subsequent to these systems, there have been a number of similar grammar guided genetic programming systems using derivation trees from a grammar as the representation for individuals. Gruau presented some strong arguments for the use of context-free grammars as a tool to set language bias on individuals. His resultant grammar guided genetic programming system is very similar to Whigham's CFG-GP. However, no depth was used to restrict the size of program, potentially leading to code bloat problems. Keijzer and Ratle and Sebag introduced a grammar guided genetic programming system similar to Whigham's CFG-GP, but with a different initialisation process, to solve some industrial problems, the resultant system was known as dimensionally-aware genetic programming. Some recent variants of these approaches were also implemented in specific programming or web languages, with the CFG being represented in Backus-Naur Form (BNF), by Tanev, MacCallum, and Rodrigues & Pozo introduced Automatically Defined Functions (ADFs) into GGGP.

2.2.2 Alternative Grammar Representations

2.2.2.1 CFG Extensions GGGP systems using CFGs are the commonest, and in some sense the most basic, form. However there has been a wide range of grammar representations used. In many, the major extension is to use a more expressive grammar representation, so as to represent a wider class of problem spaces than Context Free Languages (CFLs) – although it is noticeable that few of the problems studied with these systems actually go beyond CFLs. Of course, the most notable of these, in terms of both research priority and breadth of practical application, is the LOGENPRO system previously discussed. Others include Bruhn and Schultz’ CFGs with linear constraints; Hussain & Browse’s and Zvada & Vanyi’s attribute grammars; and Ross’ Definite Clause Translation Grammars (DCTG) - an extension of the DCG grammars used in Wong and Leung’s LOGENPRO system to incorporate semantics.

2.2.2.2 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 their introduction in the 1970s by Joshi et al. 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 and tree-structured TAG representation.

### 2.2.3 Linearised GGGP

Linear representations offer a number of seductive advantages over tree-based representations, and 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 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, machine coded GP and stack-based GP. 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 the phenotype is the derivation tree of the grammar.

In the earliest work of Keller & Banzhaf, Paterson & Livesey and Freeman, the genotype was a fixed string used to encode the indices for derivation rules in G. In these methods, the translation from a genotype to a phenotype is carried out from left to right, and the phenotype (G derivation tree) is built correspondingly. At each step, if there is still an uncompleted branch in the phenotype 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 phenotype. If the phenotype 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 phenotype is still incomplete, some random or default sub-derivation trees will be used to complete the phenotype. 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). GE is an extension of the GGGP systems with linear representation described above. Three innovations were incorporated in GE: variable length, redundancy using the MOD rule, and the wrapping operation. The chromosome in GE has variable length rather than being fixed. Each gene is an 8-bit binary number, 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 bring the key within range of the number of rules with the given LHS. The wrapping operation is used when the translation from genotype to phenotype has run out of genes while the phenotype 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 low fitness.

Since it was proposed, there has been a wide range of on-going research to develop, extend, and apply GE in many ways, including study of the effect of GE crossover on the phenotype, alternatives to the MOD rule in genotype-phenotype translation, different search strategies new representations based on the GE representation aiming to reduce the effects of positional dependence, implementation of GAs through GE using an attribute grammar, and so on.

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 very low 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.

A second issue with GE is that it does not fulfil the causality principle, that small changes in genotype should result in small changes in phenotype. Although it is easy to define GE operators that make small (bounded) and controllable changes on the genotype, the resulting changes on the phenotype are unpredictable (and uncontrollable). A small change in a gene at one position may completely change the expressiveness (coding or non-coding), or even the meaning (if there is more than one nonterminal in the grammar) of all the genes following that position. In the extreme, it may change the corresponding phenotype from feasible to infeasible.

2.3 Estimation of Distribution Algorithms using Grammars

If grammar representations have a substantial presence in GP, it is in the extension of Estimation of Distribution Algorithms (EDAs) to GP problem spaces that grammar representations have become predominant. EDAs, and the closely related ant algorithms, rely on a probability model to record the progress of evolution. Stochastic grammars have shown themselves to be an excellent candidate for this probability model.

2.3.1 Background

2.3.1.1 Linear Estimation of Distribution Algorithms

One common theory to explain the behaviour of GA, as well as GP, is that through the genetic operations - especially crossover - building blocks, which are high quality partial solutions, are discovered and combined to form better solutions. The underlying assumption of EDA is that, if we can learn these building blocks directly, instead of applying the semi-blind genetic operators (which may destroy building blocks) we may hope to significantly improve the performance of GA and GP; a probabilistic model of the set of solutions may also, in many cases, be more useful than a single solution.

EDA builds upon this idea. It uses a probabilistic model to estimate the distribution of promising solutions, and thus to guide further exploration of the search space. Through iteratively building and sampling this probabilistic model, the distribution of good solutions is (hopefully) approximated. Due to the lack of knowledge of the true distribution, we have to introduce a probabilistic model to approximate the distribution. The form of probabilistic model is strongly related to specific assumptions about building blocks.

EDAs work as follows. Assume $Z$ is the vector of variables we are interested in. $D^H(Z)$ is the probability distribution of individuals whose fitnesses exceed some threshold $H$. If we know $D^{H_{opt}}(Z)$ for the optimal fitness $H_{opt}$, we can find a solution by simply drawing a sample from this distribution. However, usually we do not know this distribution. Hence we start from a uniform distribution. In the commonest form of the algorithm, we generate a population $P$ with $n$ individuals, and then select a set of good individuals $G$ from $P$. Since $G$ contains only selected individuals, it represents that subset of the search space which is worth further investigation. We now estimate a probabilistic model $M(\xi, \theta)$ from $G$. $\xi$ is the structure of the probabilistic model $M$ while $\theta$ is the associated parameter vector. With this model $M$, we can obtain an approximation $D^{H_{opt}}(Z)$ of the true distribution $D^{H_{opt}}(Z)$. To further explore the search space, we sample distribution $D^{H_{opt}}(Z)$, and the new samples are then re-integrated into population $P$ by some replacement mechanism. This starts a new iteration.

There are many variants of EDA. For example, each iteration may create a new population, or may simply replace part of the old population with newly generated individuals. The system may learn a model from scratch or simply update the previous model.

In the field of EDA, because we have no prior knowledge of the true distribution, we have to assume that the distribution follows some well-studied model. Therefore, the accuracy of the model we choose with respect
to the true model is a vital part in the EDA algorithm. That is the reason that current EDA research is largely devoted to finding appropriate models. It is one of the natural ways to differentiate EDA methods, i.e. with respect to their probabilistic models. Most EDA research is concerned with linear string methods similar to those used in Genetic Algorithms. Since our concern is with the use of grammars in GP search spaces, we will concentrate on that small body of research which has explored these spaces.

2.3.1.2 PIPE and related non-Grammar EDA-GP

The earliest application of EDA to GP was actually very early in the history of EDA: Probabilistic Incremental Program Evolution (PIPE)\(^67\). PIPE is heavily motivated by the corresponding work in conventional EDA. Further studies based on the probabilistic model proposed in PIPE have followed, notably ECGP\(^68\) and EDP\(^69\).

Interestingly, PIPE and related works based on GP can fit very directly into the same framework as conventional EDA. The prototype tree of PIPE is a model assuming independence among random variables, so that the tree structure has no effect on the probability estimation phase of the EDA algorithm (of course, it does affect the sampling phase). EDP extends this to pairwise dependences, while ECGP extends the dependency model to multivariate dependence.

PIPE uses the Probabilistic Prototype Tree (PPT) to represent the probability distribution of tree form programs. The PPT is a full tree of maximum arity and maximum depth. Each node in the tree holds a probability table containing probabilities for all the symbols of the language.

The algorithm is essentially that of PBIL\(^64\). PIPE iteratively generates successive populations of tree-form programs using the distribution generated by the PPT. Starting at the tree root, PIPE iteratively generates individual trees by sampling the possible symbols according to the probabilities at the corresponding nodes of the PPT. Having sampled the distribution in this way, it then uses truncation selection to select a sub-population, and then updates the probabilities of the PPT to increase the probability of generating this sub-population. Thus PIPE can be regarded as a form of PBIL in which the evaluation function is built up by treating the separate genes as generating a tree structure, and then evaluating this tree structure. PIPE implicitly assumes that GP building blocks are position-dependent, i.e. in PPT, unlike most GP systems, the useful subtrees/building blocks are attached to specific positions, and cannot be easily moved to other positions.

Estimation of Distribution Programming (EDP)\(^69\) extends PIPE by modeling the conditional dependency between adjacent nodes in the PPT. Yanai and Iba argue that strong dependence is likely to exist between a node and its parent, grandparent and sibling nodes. For computational cost reasons, the implemented system actually restricts consideration to pairwise parent-child dependencies.

ECGP\(^68\) further extends the PIPE modeling, applying the approaches from ECGA\(^70\) to a tree representation. The probabilistic model is based on the PPT. Each node of the PPT is a random variable. Marginal product models (MPMs) are used to model the population of genetic programming. MPMs are formed as a product of marginal distributions on a partition of the random variables. For example, in ECGA, an MPM \([1,3][2][4]\), for a four-bit problem represents that the first and third genes have intensive interaction and the second and fourth are independent. ECGP partitions the PPT into subtrees, and the MPM factorises the joint probabilities of nodes of the PPT to a product of marginal distributions on this partition. The optimal MPM is found by a greedy search using a Minimum Measurement Length heuristic; since the partitions are subtrees, it appears that the search operator is spatial extension of a subtree to include a neighbouring node.

PPT-based EDA-GP systems are able to effectively represent a population of GP-like solution trees. The PPT model is closely analogous to the underlying tree representation, and thus provides a clear analogy to GP. However it is not so clear that there is a close correspondence between the evolution of GP building blocks and the learning which occurs in a PPT-based EDA system. What is explicitly learnt in a PPT-based EDA system is subtrees in particular locations in the PPT tree (of size 1 in PIPE, size 2 in EDP, variable size
in ECGP). This corresponds relatively closely to the rooted schemata of Rosca\textsuperscript{71} and Poli\textsuperscript{72,73}. However these rooted schemata were chosen largely because they are amenable to mathematical treatment, and can yield exact schema theorems as opposed to the inexact schema theorems which result from the non-rooted schemata of O’Reilly\textsuperscript{74} and of Whigham\textsuperscript{75}, rather than because they capture the entire effects of building-blocks in the behaviour of GP. In fact, the success in GP of modularisation methods such as Automatically Defined Functions\textsuperscript{76} and Automatically Defined Macros\textsuperscript{77} demonstrate the importance in GP of relocation and replication of building blocks. These effects cannot be captured in PPT-based models in their present forms. In the current PPT-based models, each building block must be learnt in a particular location; if a similar structure occurs in a different location in the PPT, it must be learnt separately. They have lost the relocation and replication power of crossover (and for this reason, we would predict that crossover would make a useful local search operator in PPT-based GP systems).

### 2.3.1.3 Ant-Based GP

At least in the context of GP, ant-based algorithms and EDA algorithms are very closely related, differing usually only in their probability update policies (statistically based in EDAs, biologically based on pheromone deposition and evaporation in ant algorithms). Parallel to the EDA-GP developments, a number of ant-based GP algorithms have been proposed in recent years. We consider the non-grammar variants here. The earliest application known to us is Roux and Fonlupt’s Ant Programming\textsuperscript{78}. Although the terminology is different, Ant Programming appears to us very similar to PIPE, differing only in the update policy for the probability tables. Two subsequent ant-based algorithms, Boryczka and Czech’s Ant Colony Programming (ACP)\textsuperscript{79} and Green et al’s Ant Colony Optimisation (ACO)\textsuperscript{80} use predefined grids labeled with GP symbols as probability models, with ant algorithms being used to determine paths representing good programs. In Ant Colony Programming, one variant has a single fully connected graph node per language symbol, with the concomitant problem that solutions requiring the same symbol to be used multiple times with different arguments will lead to convergence problems. A variant permits multiple occurrences of language symbols; it is not clear from our reading whether these graph structures are initialised by hand or randomly created. ACO uses a randomly initialized graph with a chosen level of symbol redundancy and connectivity. Both approaches rely on the predefined graph to determine the dependency relationships between symbols, and intrinsically can only encode pairwise dependency relationships at any given location. Rojas and Bentley’s Grid Ant Colony Programming\textsuperscript{81} resembles ACP and ACO in some ways, but introduces a temporal index, corresponding closely to the depth constraint of some of the grammar-based systems described below.

### 2.3.2 Grammar-based GP-EDAs

Grammar-based GP EDAs use some form of grammar as the probabilistic model on which the distribution learning is based. The commonest grammar form is a stochastic CFG (SCFG), in which each grammar production has an attached probability; when the grammar is used to generate a tree, productions are used stochastically, driven by the attached probabilities. Grammar-based EDAs fall naturally into two camps, EDAs using a fixed grammar as the probability model, with only the probabilities changing (analogous with PIPE) and EDAs in which the grammar model, as well as the probabilities, change as search progresses. Grammar-based GP EDA was prefigured in Whigham’s grammar learning\textsuperscript{82}. It was originally described from the then-dominant perspective of a GP system, as a grammar-based GP system with a non-uniform (learnt) probability distribution over the grammar productions, with inductive modification of the grammar during search. From an EDA perspective, it could just as readily be described as an SCFG-based EDA, with genetic local search, and ad-hoc grammar model update. Tanev’s system\textsuperscript{83} is similarly described as GP with an added Stochastic Context Sensitive Grammar (SCSG)
learning component, but can also be regarded as a grammar-based EDA with contextual structure learning similar to that of PEEL.\textsuperscript{84} However the first avowedly EDA-based GP work was the Stochastic Grammar Genetic Programming (SG-GP) of Ratle and Sebag\textsuperscript{85,86}. The model used in SG-GP is an SCFG, initialised to a uniform distribution over the productions for each nonterminal. SG-GP uses both positive and negative reinforcement, increasing the probabilities used to produce fit individuals, and decreasing those used to produce unfit individuals. SG-GP came in two variants, called the scalar and vectorial versions. In the scalar version, there is a single probability table for the grammar. However this version suffers from similar convergence problems to the first version of ACP described above: that the initial grammar may not be sufficiently complex to record the structure of the problem solution, leading to convergence problems as different parts of good solutions impose countervailing pressures on the probability values for a given nonterminal. SG-GP alleviates this problem by imposing a partial positional constraint: in vectorial SG-GP, there is a depth dimension to the probability distribution. In SG-GP, this is implemented by attaching a vector (of length equal to the maximum permitted program depth) to each grammar production. The elements of this vector represent the probabilities of applying the given production at the corresponding depths in the derivation tree. Thus a given rule may be applied with high probability at one depth in the derivation tree, but with a low probability at another. Thus vectorial SG-GP has partial positional dependence (resembling PIPE in this respect), but the positional dependence is only on depth, not on breadth: building blocks may (indeed must) be shared between nodes at the same depth.

A remarkably similar mechanism was used as the basis of the (independently-derived) Program Evolution with Explicit Learning (PEEL)\textsuperscript{84}, which also uses SCFGs with a depth constraint – though described in that publication as a stochastic parametric L-System. However PEEL also adds a location constraint – initially empty (hence equivalent to SG-GP), but gradually evolving to incorporate a context (ie parent and possibly deeper ancestor nodes) in which the production may be applied. In this way, PEEL’s initial bias is to share building blocks at a given depth, but it may learn that particular building blocks are not to be shared. The context learning algorithm relies upon a heuristic that detects when particular nonterminals are both highly influential and poorly converged. The underlying assumption is that if a nonterminal is heavily used, yet its probability does not converge, it is likely that there are conflicting uses of the nonterminal which require different probability distributions, and that these conflicts may be resolved by determining the context in which the nonterminal is used.

Bosman and de Jong\textsuperscript{87} also make use of an SCFG with a depth constraint, independently deriving a very similar approach to SG-GP and PEEL (but with additional complications arising from maintaining as the population only the derived sentences of the grammar as in Antonisse\textsuperscript{10}, rather than the derivation trees as in many later GGGP systems – this imposes the computational cost of parsing on each generation, and requires grammar restrictions to avoid ambiguous grammars). Like PEEL, Bosman and de Jong learn the structure of the grammar as well as the probability distribution, using greedy search with rule expansion as the search operator, and a Minimum Description Length (MDL) metric as the search heuristic.

A rather different SCFG-based system is Generalised Ant Programming (GAP)\textsuperscript{89}. GAP records the whole path an ant has visited, building path-based rather than content-based transition tables, resulting in a grammar-based system somewhat resembling EDP (in that the probabilities represent pairwise interactions between nodes).

Ant-TAG\textsuperscript{90} resembles scalar SG-GP in using only a single grammar probability table, but uses a stochastic TAG grammar rather than an SCFG as the probability model. Possibly as a consequence, it was not particularly successful as a pure system, but gave good results when local search based on the crossover operator was added.

Grammar Model-based Program Evolution (GMPE)\textsuperscript{91} takes a different approach. Whereas the
structure-learning systems described above learn the grammar model incrementally, using ad-hoc heuristics to expand the grammar, GMPE uses standard grammar learning methods from NLP, and learns the grammar model anew from the selected population each generation. It starts from a very specific grammar, in which the rules directly describe the selected population, and uses stochastic hill-climbing search, with rule merging as the sole operator and a carefully-derived Minimum Measurement Length (MML) metric as the search metric. GMPE has demonstrated extraordinarily good search performance in terms of numbers of fitness evaluations required to find a solution, but is not suitable for application to practical problems because of the very high computational overhead embodied in the grammar learning algorithm.

GMPE’s computational cost arises because complex MML calculations are required for each merge, to ensure that the merge does not over-generalise. A very recent, and less conservative, variant – simple GMPE (sGMPE)\textsuperscript{92} – relies on the observation that the outer EDA algorithm is a specializing algorithm (the specialization resulting from the selection step), so that a small degree of over-generalisation in the learning phase can be tolerated, so long as it is not sufficiently large as to counteract the specialization resulting from selection (which would lead to stagnation). Some types of GMPE merges (unifications) will always lead to a reduction in MML, since they do not cause any generalization. sGMPE first deterministically carries out all available unifications, then stochastically applies a user-tunable number of generalising merges, the idea being that the user tunes the number of merges to give satisfactory performance without over-generalisation.

sGMPE has delivered a slight performance improvement over GMPE in terms of number of evaluations per solution, but a halving of the computational cost of the learning phase, at the cost of adding an extra tuning parameter to the algorithm; performance is relatively robust to changes in the tuning parameter.

3 What do Grammars Bring to GP?

The important place of grammars in modern GP suggests that grammars bring some valuable benefits; however there are also costs associated with these benefits. What are these benefits and costs, and how do they affect grammar-based GP systems? We consider, first, properties which apply to most or all grammar representations, and then some specific properties which are limited to only a few representations.

3.1 Properties of Grammars in General

3.1.1 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 primary benefit of restricting the search space is, of course, 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 serious, that the solution may be isolated by the grammar constraints and may be difficult to reach.

One of the commonest uses of grammars in GP applications has been to impose type restrictions; when used in this way, it is equivalent to Strongly Typed Genetic Programming (STGP)\textsuperscript{15}.

Another common use 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 input. 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.
3.1.2 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.

3.1.3 Homologous Operators
To the extent that the grammar used reflects the semantic structure of the domain, the homologous operators provided by GGGP replace one component of a solution with another with similar function. This homology may be expected to result in more effective search.

3.1.4 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.

3.1.5 Opportunities for Search Space Transformation
Since the publication of the messy GA⁹, 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. All rely on the restrictions imposed by the grammar rules to impose 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 come at a cost. For one thing, the operators are no longer homologous after the linearization transformations. In all these transformations, an alteration early in the genotype often changes the interpretation of the subsequent genotype, so that (depending on the grammar) the proportion of highly disruptive operators may be much higher than in the underlying tree-based representation.

3.1.6 Feasibility Constraints
Grammar-based approaches also bring with them some disadvantages. As with standard GP, grammar guided genetic programming is still far from resembling GA in its algorithmic flexibility, despite some significant efforts in this direction. In tree-based GGGP systems, it is even more difficult than in GP to design operators, especially those making small local changes to the genotype. In addition to the constraints imposed by the GP tree grammar, the derivation trees in GGGP, are even more constrained by the rewrite rules of the formalism. For GGGP systems with linear representation (including, we believe, the recent extensions of GE), the problem is partly solved in the sense that it is easy to design new operators, including local operators, in the genotype space. However, the non-causality in the genotype-phenotype mapping means that this effect disappears in the phenotype space.

3.1.7 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. As a result, 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.

3.2 Properties of Specific Grammar Formalisms

3.2.1 Incorporating Semantics
A number of GGGP systems – Ross’ DCTG-based system\(^1\) and attribute-grammar systems\(^{38,39,40}\) – 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. The latter is generally the only programming required, and may typically be less than ten lines of code. In incremental learning, where the grammar and fitness function may be altered by the user to guide search, it is often the case than no re-programming is required at all.

3.2.2 Operators
As discussed above, defining new operators – including local search operators and biologically motivated operators – is extremely difficult in standard 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 rescaling resulting from the linearization transformation means that the effects in the phenotype space may be very different. One of the key advantages claimed for TAG-based grammar systems\(^{46}\) is that the ease with which new operators may be defined, 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.

3.2.3 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 phenotype space (CFG derivation trees) in a controlled way corresponding to the structure of the grammar (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 Conclusions
Grammars offer many advantages in Genetic Programming, and have been widely used for this purpose. The field is still developing rapidly, with many important research directions currently being explored. Perhaps the key areas are in representation and in search. In representation, there is important current work on alternative linearization transformations, and on alternative grammar representations. In alternative search algorithms (ie EDA and ant-based algorithms), grammar representations offer clear advantages, with the result that grammar-based systems form one of the main threads in this field.

References


27 M. L. Wong, personal communication, 2003


92 Y Shan, Program Distribution Estimation with Grammar Models, PhD Thesis, University of New South Wales at the Australian Defence Force Academy, Canberra, Australia, 2005