Genet Program Evolvable Mach (2014) 15:115–167
DOI 10.1007/s10710-013-9205-x

Probabilistic model building in genetic programming: a critical review

Kangil Kim · Yin Shan · Xuan Hoai Nguyen · R. I. McKay

Received: 9 December 2012 / Revised: 5 June 2013 / Published online: 17 September 2013

Abstract Probabilistic model-building algorithms (PMBA), a subset of evolutionary algorithms, have been successful in solving complex problems, in addition providing analytical information about the distribution of fit individuals. Most PMBA work has concentrated on the string representation used in typical genetic algorithms. A smaller body of work has aimed to apply the useful concepts of PMBA to genetic programming (GP), mostly concentrating on tree representation. Unfortunately, the latter research has been sporadically carried out, and reported in several different research streams, limiting substantial communication and discussion. In this paper, we aim to provide a critical review of previous applications of PMBA and related methods in GP research, to facilitate more vital communication. We illustrate the current state of research in applying PMBA to GP, noting important perspectives. We use these to categorise practical PMBA models for GP, and describe the main varieties on this basis.

Keywords Probabilistic model building · Estimation of distribution · Ant colony · Genetic programming · Iterated density estimation · Prototype tree · Stochastic grammar

K. Kim · R. I. McKay (✉)
Structural Complexity Laboratory, Department of Computer Science and Engineering, Seoul National University, Seoul, Korea
e-mail: rimsnucse@gmail.com
URL: https://sc.snu.ac.kr

Y. Shan
Australian Government Department of Human Services, Canberra, Australia

X. H. Nguyen
Hanoi University, Hanoi, VietNam

1 Introduction

Evolutionary Computation (EC), has become a widely used meta-heuristics optimisation and learning technique, with a wide variety of applications in engineering, medicine, humanities etc. The primary inspiration for this field arises from natural phenomena, especially biological evolution. Perhaps the best known form is the Genetic Algorithm (GA), which applies nature's evolutionary process to problem solving, using selection, crossover, and mutation. The core process maintains a population of possible solutions (individuals), and evaluates their fitness as a solution (fitness to the environment). Fitter solutions probabilistically survive selection, and are then adapted into new solutions by combining or mutating them (reproduction). However EC covers a much wider range of techniques than this basic concept.

In this paper, our discussion will broadly cover Probabilistic Model-Building Algorithms (PMBA)—algorithms that generate probabilistic models to guide their search. PMBAs fall into a number of forms. One form, modelled on natural processes, has led to the field of Ant Colony Optimisation (ACO [21]). A second, motivated more by statistical theory, is known by a number of names: Estimation of Distribution Algorithms (EDA [49, 67]), Probabilistic Model-Building Genetic Algorithms (PMB-GA [84, 85]) and Iterated Density Estimation Evolutionary Algorithms (IDEA [15]). To cover all such algorithms, including extensions to different problem domains, we will use the terminology 'PMBA' except where we need to distinguish a specific form, when we will generally use the terms ACO and EDA unless more specificity is required.

The basic algorithm resembles a conventional GA except that it replaces crossover and mutation with twin processes of sampling and model update. Sampling is used to generate new individuals, whose fitness is evaluated. Fitness is used to weight the samples; usually a Boolean weighting (i.e. truncation selection) is used, but more sophisticated weighting schemes are possible. The model is updated (or explicitly learnt—some systems ignore the previous model and learn a new model *de novo*) from this selected sample. PMBAs have several interesting characteristics—notably improvements in performance as problem solvers and ease of statistical analysis of behaviour.

Genetic programming (GP) extends the evolutionary paradigm to search spaces consisting of function expressions or programs. Prima facie, the use of probabilistic models is appealing in GP, since the basic algorithm structure is almost the same—the biggest difference is in representation; standard GA uses fixed string individuals, while GP uses variable, often tree-structured, individuals. The tree representation requires more complex operators and generates a more complex search space, however it has been successful in solving many problems with tree structured information [36, 107].

Practically, a number of researchers have applied PMBAs to GP representation.[1] However work in this area has been somewhat sporadic, probably because it is quite

[1] In Shan's survey [107] they were referred to as EDA-GP; we prefer to use the more general terminology PMB-GP, reserving EDA-GP for systems more closely modelled on EDA.

complex to learn tree-structured representations with the conventional Graphical Models (GM [45]), which have formed the backbone of most PMBAs [50].

A critical review of the field is needed for a number of reasons. Most important, researchers have not always been fully aware of other work, leading to inefficient research advancement in this field. While Shan's survey went some way to overcoming this, it is now rather dated, and its circulation has been restricted. The recent review by Larranaga et al. [50] of probabilistic graphical models in evolutionary computation briefly noted EDA-GP, but did not cover the field in detail.

While providing an up-to-date guide to research in PMB-GP and related areas, this survey also aims to delineate the current position of PMB-GPs relative to other methods, and to provide a guide for researchers outside the PMB-GP streams. We will also try to elucidate some important issues for future research, in particular the values of different representations, and at a more general level, the roles of positional determinacy, context dependence and structure learning. Our intended audience is practitioners in evolutionary computation, but particularly those entering the field of PMB-GP, those familiar with PMB systems who are interested to see how they are used in GP, and conversely those familiar with GP who are interested to see how PMB methods are currently being used, and what the problems and issues may be.

To clarify the range of PMB-GP, in this paper we aim to cover systems that learn probabilistic models for evolutionary search in the domains generally accepted as covered by genetic programming. As in optimisation, not all PMB-GP systems are expressly based on EDA. There has been a range of studies in other research streams, such as Ant Colony Optimisation (ACO) for GP [1, 8, 10, 11, 12, 25, 42, 72, 93, 96] and Grammar Induction (GI) [13, 114, 120].

Figure 1 may help to clarify our terminology. At the first level, the terminology distinguishes by problem domain because this enormously affects the issues under consideration. The second level distinguishes based on the use of the probabilistic model, primarily between EDA and ACO. The question marks signify that we do not intend the terminologies to be exhaustive, but simply to recognise important distinctions among existing systems. For example, at the first level, search spaces such as permutations are arguably sufficiently different from the Euclidean spaces of classical PMB-GA to fall under PMB-?, while at the second level, hybrid systems such as CMA-ES might well form a third category under PMB-GA. Since these are outside the primary focus of the paper, we do not consider these issues further.

The review is organised as follows. We first introduce the basic concepts of PMBA, and specifically of EDA and ACO in Sect. 2. Section 3 provides some

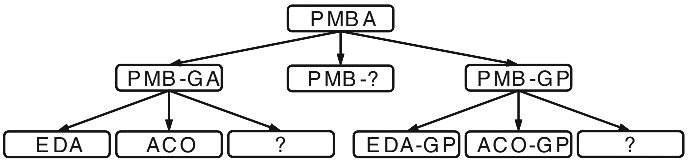

Fig. 1 Hierarchical Terminology for PMB systems

illustrative examples of EDA-GP systems to set the scene. To clarify the relationships between the diverse strands of PMB-GP research, we briefly preview current PMB-GP model representations in Sect. 4. Extending beyond these superficial issues, we examine semantic issues in learning a model from GP representation, inspired by an analysis of conventional models in Sect. 5. Based on these perspectives, we categorise and analyse current PMB-GP models, providing a brief introduction to their mechanisms, in Sect. 6. We provide a brief review of research directions and challenges in PMB-GP in Sect. 7, and summarise in Sect. 8.

2 Probabilistic model-building algorithms

PMBAs are evolutionary algorithms, learning a probability distribution over individuals. Though often not explicitly stated, the underlying aim is, in the limit, to converge to a degenerate distribution that describes the optimal solutions to the problem. For practical reasons (symmetry breaking—so that not all solutions will be retained; retaining exploration—so that full degeneracy is avoided) this target is usually adjusted in practical systems.

The search spaces are generally either infinite, or extremely large, so that exhaustive and random search are infeasible. Thus assumptions are needed, both to guide the search, and to permit economical description of the current state of the search. Almost all non-random search methods rely on an underlying distance metric on the space, and on the assumption that there is some correlation between differences in fitness between points and the distance between them. PMBAs generally rely on one further assumption: that an individual consists of components, and that there is a correlation between the fitness of individuals sharing a component. Thus PMBA models represent these components, and attempt to relate them to fitness. Specific systems parameterise these models in different ways, imposing further assumptions (unimodality, symmetry etc.) on the structure of the search space.

Typical PMB algorithms have the form shown in algorithm 1:

Algorithm 1 A Typical PMB Algorithm

```
Generate N individuals randomly from an initial probability model
while not (termination condition is satisfied) do
    Evaluate individuals using fitness function
    Select the best individuals
    Update the stochastic model using the selected individuals
    Sample new individuals from the model distribution
end while
```

This algorithm omits the step of the conventional GA which applies variation operators (crossover and mutation) to individuals, replacing it with a model update and sampling process, which can be viewed as a generalisation of crossover to recombinations of the whole population. The hope is that, if the models are appropriately expressive, all relevant combinations can be sampled. Thus if the model representation is suitable to the problem, PMBAs may avoid loss of important building blocks during evolution. This was a key impetus for EDA work in particular.

Beyond this motivation, PMBAs offer other advantages over more conventional methods as problem solvers and analysis tools. The probability distribution collected at each generation provides substantial information, which can influence the next state of the PMBA system, even if it is only approximated. Compared to conventional evolutionary approaches, it can lead to much easier analysis. For example, many researchers have analysed the convergence of EDAs [49, 110, 111].

The probability distribution offers one further advantage over conventional evolutionary algorithms: the final product is not merely a single solution, but a probabilistic description of a space of good solutions. While this has limited importance in standard PMBA work, it potentially offers important advantages in PMB-GP, particularly in helping to generate explanatory models.

Many PMBAs have been proposed, with a wide variety of operators and model structures. For example, the hottest issue in EDA research to date has been constructing models which have substantial representation power for complex dependencies. More exact representation of dependencies, particularly in complex fitness landscapes, has led to distinct improvements in performance [49].

2.1 Ant colony versus estimation of distribution

We have noted the close relationship between EDAs and ACO: they have a similar ultimate target, and they both aim to get there by probability models. So how do they differ? EDAs generally aim to estimate, at each generation, the unique distribution best justified by the combination of the previous model as prior, and the current data (the selected population). ACOs aim to converge to the same final point, but with less concern about the justification of the intermediate models, relying on the iterative process to converge, in the end, to the right distribution. However this distinction is less clear-cut than might at first appear, for two main reasons. For more complex problems, it is generally not feasible for an EDA to construct a single best-justified posterior model, it can only be approximated. Equally important, doing so in each generation is likely to amplify stochastic bias, so most EDAs relax the learning rate in ways that are not too different from the pheromone evaporation of ACOs. Thus in the end, the differences turn out to be more of emphasis than of fundamentals.

2.2 Model construction

Most learning mechanisms used in EDA originate from probabilistic Graphical Models (GM) [45], a well-studied machine learning technique. The best-known are Bayesian Networks (BN) and Markov Networks. EDAs often use a BN represented as a Directed Acyclic Graph (DAG). Each node represents a random variable, with the directed edges indicating dependence relations between them. Within this framework, different EDAs use a range of methods to represent the probability distribution. The earliest systems used independent variables, while more recent versions can represent multivariate dependencies. For a comprehensive recent survey of probabilistic Graphical Models for EDAs, readers are referred to [50].

The individual representation is usually a fixed linear string, because EDA typically aim to solve the same classes of problems as GA . Each variable is mapped to a position of the string, thus a variable observes the values at a specific location in all individuals in a population.

ACO methods, which originated in combinatorial problems, more typically impose a specific dependence structure (usually imposed by the problem) both in sampling and learning stages. In these methods, there is generally less emphasis on learning linkages between variables.

3 Building models with genetic programming: examples

In this section, we describe how two classic PMB-GP systems construct models. They are Probabilistic Incremental Program Evolution (PIPE) and Stochastic Grammar based genetic programming (SG-GP), two simple (and early) PMB-GP systems which form the basis of many more recent systems.

3.1 Probabilistic incremental program evolution

PIPE, the first fully-developed PMB-GP system, introduced the widely-used Probabilistic Prototype Tree (PPT) as a model. We discuss its properties in detail in Sect. 6. PIPE omitted GP's population reproduction stage, in which variation operators are applied. It replaced this with two stages, of learning a distribution (represented by a PPT), and generating individuals from it.

3.1.1 Model structure: probabilistic prototype tree

PIPE's choice of model structure is perhaps the most obvious: using a tree-structured model to represent distributions over tree-structured individuals. Figure 2 shows a simple example of a PPT for a problem in a domain using arithmetic and other operators: $+$, $-$, $\times$, %, sin, cos, exp, log, random constants (R) and a single variable (s). Since the maximum arity among these operators is 2, the PPT is a full binary tree, as seen on the left of the figure (more generally, if the problem domain symbols have maximum arity n, then the PPT is a full binary tree of arity n). Each node of this tree represents a random variable (in this case, with an independent multinomial distribution) ranging over the symbol set of the problem; initially, they are uniformly distributed, but their probabilities are updated from observations.

Figure 2 illustrates the PIPE sampling process. PIPE starts at the root of the PPT (left side of the figure), sampling an outcome which is used as the root value in the individual (right side of the figure). The process then recursively visits the children of the sampled node, depending on the sampled value: if a particular argument is

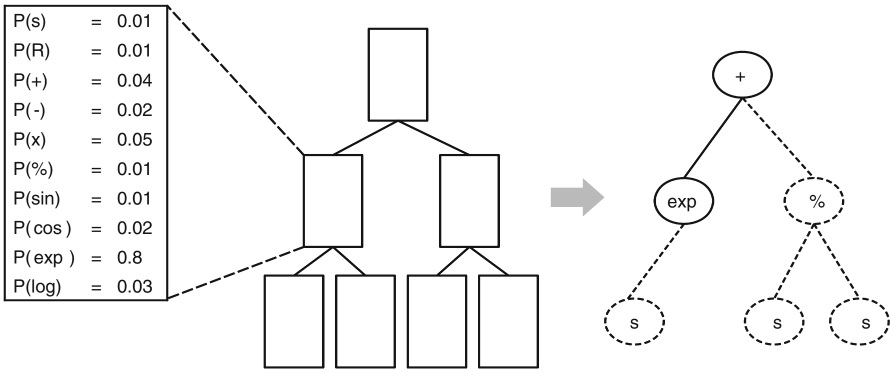

Fig. 2 PPT Model Example (Left : PIPE PPT model, Right : Sampling a Tree Individual from the Model

defined (the left arguments of sin, cos, exp, log, both arguments of $+$, $-$, $\times$, %), the corresponding child node in the PPT is sampled. In other words, PIPE uses ancestral sampling, generating an outcome and visiting child variables repeatedly until it generates a completed tree individual.

In the learning process, the PPT nodes are mapped to corresponding positions in all (selected) individuals. The resulting frequency table at each PPT node is updated by the frequency of components in the selected good individuals, using a variety of update strategies.

3.1.2 Precursors and extensions

Despite PIPE's simplicity, it has inspired much subsequent research. PIPE may be considered as a natural application of the ideas of Population-Based Incremental Learning (PBIL) [4] and Univariate Model Distribution Algorithm (UMDA) [66] to GP representation. followed by subsequent models representing dependencies between variables, so PIPE has been followed by subsequent models representing dependencies between the random variables attached to tree locations. Estimation of Distribution Programming (EDP) [129–132] adopted bivariate conditionally dependent variables (children depending on parents) in place of independent variables. Extended Compact genetic programming (ECGP) [104] used multivariate conditionally dependent variables, as did Program Optimization with Linkage Estimation (POLE) [31, 33]. These had been foreshadowed to some extent by Handley's [28], which used a DAG to compress the representation of the GP population (considered as a forest). However there is no evidence in his published work that Handley built a probability model on this DAG or used it for sampling.

Other potential variations in these systems include changes to the learning and sampling mechanisms. For example, some variants of PIPE use incremental learning with a pre-specified learning rate.

A common theme among these extended methods is the mechanism for handling GP constraints, so as to generate only feasible individuals. Another is the mapping strategy between observable locations and random variables; this will be discussed in more detail in Sect. 6.

3.2 Stochastic grammar based genetic programming

SG-GP was another early PMB-GP. Grammars provide a well known approach to handle search space restriction and guarantee the feasibility of GP individuals [120]. It is relatively straightforward to extend grammar-based GP to a PMB-GP, using probabilistic grammars as the probability model. Specifically, SG-GP uses Stochastic Context Free Grammars (SCFGs) as models.

3.2.1 Model structure: stochastic context free grammar

An SCFG is a straightforward extension of Context Free Grammars, adding weights to each production. The weights are normalised so that the weights on all productions from the same nonterminal sum to 1, and thus they can be regarded as a probability table. We may formalise this in the following definition (using the notation $(S)^*$ for the Kleene star of set S) of a stochastic grammar model M.

$$\begin{aligned}
M &= \{N, \sigma, S, R, p\}\\
N &= \{n | n \text{is a nonterminal symbols}\}\\
\sigma &= \{t | t \text{is a terminal symbol}\}\\
R &= \{r | r \text{ is a production of form } n \rightarrow \lambda, \text{ where } \lambda \in (N \cup \sigma)^*\}
\end{aligned}$$

In a production, n is known as the left-hand side (LHS), while λ is the right-hand side (RHS). $p : R \rightarrow (0, 1)$ associates each production r with a probability $p(r)$ satisfying:

$$\forall X \in N, \sum_{r \text{hasLHS} X} p(r) = 1$$

An example SCFG for symbolic regression problems is shown in Table 1. In this grammar, each LHS symbol can be viewed as a random variable with an independent multinomial distribution. Each production defines an outcome. Mapping these variables to observable locations is based on matching. If we observed Exp in a tree individual, it is regarded as an observation of the random variable Exp of the SCFG.

The learning mechanism of SG-GP is based on the frequency of occurrence of particular symbols in specific contexts. In the simplest form, learning through Maximum Likelihood Estimation (MLE), the system evaluates the frequency of each outcome, and normalises for each variable. Sampling of the SCFG starts from the grammar's start symbol. In the example, S is the start symbol. Selecting an outcome from the probability distribution for S, the SCFG can grow a tree individual by adding symbols from the RHS of the selected production as children, repeating recursively until the process terminates.

Table 1 Stochastic distribution over context free grammar for symbolic regression problems

LHS		RHS			Probability
S	→	Exp			1.00
Exp	→	Exp	Op	Exp	0.50
	→	Pre	Exp		0.25
	→	x			0.25
Op	→	+			0.10
	→	−			0.20
	→	×			0.10
	→	/			0.60
Pre	→	sin			0.25
	→	cos			0.25
	→	e^			0.25
	→	ln			0.25

3.2.2 *Extensions*

The origin of stochastic grammars in GP stemmed originally from the desire to limit the search space through type, grammar or other constraints. Examples include the strong typing of Montana [65], the constrained trees of Janikow's ACGP [37] and the grammar guided genetic programming (GGGP) of a number of authors [23, 120, 127]. Whigham [121, 123] subsequently incorporated stochastic grammar learning in a hybrid genetic system, with SG-GP following a few years later.

In building explicit variables, one direction of extension of SG-GP has followed a similar course to PPT: incorporating conditional dependency into the model. Another has extended beyond simple context free grammars to support greater expressive power in specifying the distribution. For example, depth information may be added as annotations to the LHS symbols as in vectorial SG-GP (vSG-GP) [89], so that the distribution varies with the depth. More general annotations as in PAGE [32, 34] are further extensions. In other alternatives, we may learn a completely new grammar structure as in GMPE [106], or we may extend beyond context freeness to the mildly context-sensitive Tree Adjoining Grammars [1] or other Context Sensitive Grammars (CSG) [114]. The common property of these systems is their reliance on mapping variables to observable locations based on symbol context. This contrasts with PPT models, in which the mapping is based on position. These issues will be discussed in more depth in Sect. 6.

4 An overview of PMB-GP

Over the last 10 years, there has been a surge of research in applying PMBA concepts to GP. However, the work has been presented in a number of different streams (most notably, separating between EDA and ACO streams) with limited interaction, and often even awareness, between them. These streams have shared a number of important issues, to the extent that a rational categorisation would cut across the streams and emphasise different criteria. Here, we briefly summarise the criteria we will use to structure our discussion. To a substantial degree, it is based on the representation used.

In discussing representation in PMBAs for GP, we need to take some care with terminology. There are two representations involved: the representation for individuals in the sampled and selected populations (the individual representation) and the representation used in the stochastic model (the model representation). The choice of individual representation restricts the possible model representations, but it does not determine it.

4.1 Representation of individuals

One of the core criteria for distinguishing the problem domains of evolutionary systems is the representation they use for individuals. For example, in GA systems, individuals are represented as of string of values. In classifier systems, they are represented as a set of rules. GP systems mostly use a tree with structural information, although some systems use special representation for programs, as in linear genetic programming or Cartesian genetic programming [18, 5, 62, 63]. This fundamental difference between evolutionary systems leads to important differences in their operators (crossover and mutation), and other system components. We concentrate on the differences between GA and GP systems, since these are the most informative for the extension of GA-based PMBA methodologies to GP. Figure 3 depicts typical GP evolutionary operators; by comparison with GA, these operators differ in the need to deal with structural information in addition to simple value information. When probabilistic methods are substituted for these evolutionary operators, similar issues arise. Other GP systems use different representations, for example a general graph [40, 41, 116], but the problem of dealing with additional structural information still arises.

4.2 Structural information in GP individuals

Each component of a GA or GP individual has a value (in GP, potentially a value from the set of function symbols rather than a domain value). In GP, these values

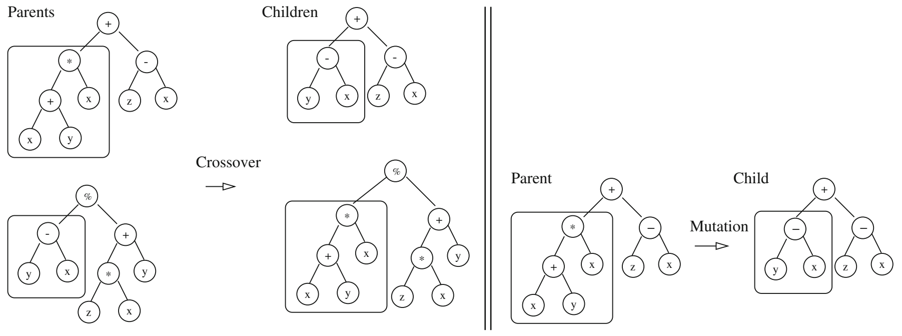

Fig. 3 Operators for GP (left: subtree crossover, right: subtree mutation)

are required to follow some domain constraints on their relationships (e.g. type constraints, constraints on the number of arguments). Information about these constraints influences evolutionary algorithms, by determining the boundary of the search space. When individuals fail to satisfy the constraints, we can omit them from the candidate solutions. This deterministic division of the search space allows us to apply some forms of human knowledge to evolutionary algorithms without the complexity and uncertainty of learning. When we are certain we know such relationships, we can use this knowledge to reduce the search space, inducing greater efficiency in GP systems [64]. The earliest GP systems such as Koza's tree-based GP [46] imposed only weak constraints such as number of arguments, but there has been a trend toward increasing the complexity of constraints in GP, from strongly-typed GP systems [64, 65], to systems imposing more general syntactic constraints [120] and even semantic constraints [77, 128].

Unfortunately, making use of this information in extending a PMBA to GP applications is far from trivial. Effectively extending ordinary evolutionary algorithms to GP was complex enough, requiring adaptation of the operators right from the start, and subsequent incorporation of mechanisms to control bloat, and to handle diversity issues that arise in different ways than in GA. When it comes to extending PMBA methods to GP representation, there are still more complex issues. The solutions proposed have been very diverse, but mostly fall into one of two groups: transforming the GP representation to a GA-like structure so as to be able to apply a near-standard PMB-GA, or conversely extend a PMBA model to apply to a more typical GP representation.

4.3 PMB-GP systems with GA-like representation

One reasonable approach to handling these problems is to use a linear GP representation, thus reducing the apparent difference between conventional PMB-GA models and the GP system's structural representation. This permits the use of standard, well-studied structures for probabilistic models, encoded linearly (perhaps extended with some structural information). However in many cases, these linear models can also generate semantically meaningless individuals. Three main strategies have been used to handle this:

1. adding constraints to the model, so infeasible individuals are not generated [17]
2. dropping infeasible individuals (i.e. a sudden-death constraint handling mechanism [91])
3. fragmenting the linear structure so that individuals can encode this relational information [88]

Most of the methods using a linear representations have adopted special processes of transformation or segmentation of individuals. There is one common thread between them: the model has no specific way to represent or distinguish the structural relationships between the components of individuals. Thus knowledge about such structural relationships, which is available to the user of the system, cannot be passed to or used by the system in its search. This is not surprising. The original focus of graphical models, the form most commonly adopted in such

PMB-GAs, was purely on attribute data—they were not designed for learning from relational data, such as is normally present in a GP population. Ignoring this relational information can lead to inefficiencies in the model learning stage of PMBAs applied to GP representations. We will discuss these issues of structural formalism further in Sect. 5.

Nevertheless, this direct application of classical graphical models can be an effective way of extending PMBA methods to GP. Simply changing the representation of individuals to a linear one is relatively straightforward. For some problem domains, we can relatively easily gain performance benefits relative to conventional GP.

To date, one of the best-known EDA methods, the Bayesian Optimization Algorithm (BOA) [83], has been extended to linear coded GP in several ways. Other methods have used an N-gram table, inspired by Natural Language Processing (NLP) [57]. Cartesian genetic programming—whose components are anyway linearly encoded with additional linkage information,—has been combined with the conventional EDA Univariate Marginal Distribution Algorithm (UMDA [66]). We will discuss details of these models in Sect. 6.

4.4 PMB-GP systems with GP-specific representation

Much PMB-GP research has concentrated on extending PMBA methods to a typical Koza-style tree GP representation. Researchers have invested a lot of effort in designing specific model representations for this form of GP, on the basis that a suitable model structure could provide a more effective means of learning from the information held in the selected population. We have already seen examples of the two main kinds, prototype trees and stochastic grammars (SG).

A number of researchers have extended PIPE's [98] simple univariate model over PPTs, attempting to learn more complicated dependencies in this underlying structure [31, 33, 104, 130, 131].

Correspondingly, there have been a number of PMB-GPs based on stochastic grammars (SG) [123]. The first such was a hybrid evolutionary/model-based approach [123], arising out of grammar-guided GP [23, 120, 123, 126, 128], though described from a genetic perspective. The first explicitly described SG PMB-GP was that of Ratle and Sebag [89], which we illustrated in Sect. 3.2. These early methods learnt univariate statistical models over a fixed grammar; many subsequent extensions have adapted the process to learn some level of grammar structure in addition to the probabilities [34, 105, 106].

In other areas of PMB-GP, very similar methods to the previously-described EDA-GP systems have been studied under the rubric of Ant Colony Optimization (ACO) [21]. ACO is based on a biological analogy to the way real ants find food resources. Traveling over a variety of routes while leaving pheromone trails, ants return to their nest following the same route if they find a food source. If this process is repeated enough, ants can find an optimal path to the food source. The motivation is very different from EDA, but the underlying model, update and sampling mechanisms are very similar [133], to the extent that the across-group similarities between specific ACO-GP and EDA-GP algorithms are often greater than within-group. In particular the preceding categorisation, based as it is on genotype representation, applies almost as well to ACO-GP systems as it does to EDA-GP.

We will use the above categorisation to organise our discussion of specific implementations in Sect. 6; but first, we examine in more detail some of the issues in choosing a stochastic model for PMB-GP

5 Issues in choosing and learning a stochastic model for PMB-GP

The two main approaches to PMB-GP we described above—transforming to a PMB structure suitable for GA-like problem spaces, or defining a specific representation for PMB-GP—raise very different issues. In the former case, they depend intimately on the specific transformation, and thus are difficult to incorporate into a general discussion. Hence we emphasise the latter here, though specific considerations may be applicable to the former as well. The core issues are how to define a suitable distribution, able to represent the regularities likely to occur in post-selection GP populations; and how (in the case of systems in which model structure is permitted to change) to learn the appropriate changes. Neither of these are trivial issues.

Although we frame our discussion around PMB-GPs with a typical expression-tree phenotype representation (thus covering the substantial majority of all PMB-GP system), the ideas discussed can be extended, in most cases, to other PMB-GP systems as well. We will allude to this when we consider individual PMB-GPs in Sect. 6.

Due to the complexity of the issues involved, there is currently no accepted, well-founded theory. Nevertheless, it is possible to observe a number of issues arising from analysis of previous systems, which might be of general relevance to PMB-GP algorithms. The underlying question is: what kinds of distribution are best suited to form the basis of the model structure?

5.1 Semantics: modularity

The fundamental goal of evolutionary search, finding good solutions, is closely related to understanding the internal structure of fit individuals, falling into the areas of linkage analysis and schema theory in GA and GP [24, 48]. Strongly related internal components or structures are known as *building blocks* [24], and are viewed by many as having critical importance in finding better solutions. Understanding such structures, especially in GP, is still a work in progress, but the work to date clearly indicates the strong influence of the semantics of problems [2, 59].

The building block hypothesis is a heavily debated issue both in the wider field of evolutionary algorithms, and in genetic programming in particular [7, 24, 48, 59, 76]. In this context, building blocks refer to subcomponents that can be used to construct good solutions; finding them is expected to yield better individuals in subsequent generations. According to schema theories based on building blocks, evolutionary algorithm performance is heavily affected by how well the algorithm detects and combines building blocks. Building blocks are seen as highly correlated with semantics, because they are the components needed to build the fittest individuals, the fitness being determined by semantics. Understanding such structures, especially in GP, is still a work in progress.

The view underlying PMBAs in general is that, if we can appropriately model these structures, either ahead of time or through explicit run-time learning, PMBAs may show better performance than conventional approaches [49]. Thus whatever may be the situation with evolutionary algorithms, PMBAs are strongly committed to some form of building block hypothesis; the differences between them lie principally in the building blocks they can represent, and the methods they use to learn them. Building blocks are even more important in PMB-GP, because building blocks are what a PMB-GP model explicitly represents. A good PMB-GP representation will thus be one which effectively captures the structure of such building blocks for a class of GP problems. In this section, we point out likely characteristics of such GP building blocks. Next we discuss the issues arising in attempting to capture them in the distribution learning of PMB-GP.

5.1.1 Modularity in GP theory

In GP, our assumptions about the behaviour of building blocks can be captured in a word, modularity [107, 108]. It refers to the flexibility of locations where building blocks are observed, combining three more detailed aspects:

1. Connectivity: Building blocks are connected substructures
2. Relocatability: Building blocks may be useful in different positions
3. Re-use: Building blocks are repeatable in an individual

Connectivity: What crossover—the dominant operator in GP search—most obviously conserves is connectivity between components, destroying only two links while conserving all others. This has inspired the recognition of a variety of connected tree substructures as building block units in GP. In particular, all forms of GP schema theory to date have used some form of connected substructure for analysis [74–76, 87, 94, 122].

Relocatability: A second important property of crossover is its ability to relocate a component. This is most obviously the case when an individual is crossed with a near-copy of itself: in most cases, the exchanged component will be moved to a new location. Recognition of this has led to various forms of automatically defined and relocatable components, which we discuss in more detail below.

While some forms of GP schema theory use fixed-location schemas [87, 94], the primary reason for this choice was to obtain exact formulae for schema propagation, and a greater similarity to GA schema theory. Earlier GP schema analyses used relocatable schemas [74, 122], and demonstrated amplification of fit schemas in these contexts, suggesting that GP can use either fixed or relocatable schemas, whichever is suited to the particular task.

Re-use: Sequences of crossovers can result in multiple copies of a building block within a single individual; most forms of automatically defined components support such re-use. Again, some forms of GP schema theory [122, 74] have incorporated re-usable schemas that can occur more than once in the same individual.

5.1.2 Modularity in GP practice

A number of GP studies have aimed to identify and retain meaningful building blocks, among them Automatically Defined Functions (ADF) [47], Genetic Library Builder (GLiB) [3], and adaptive representations [95]. The core idea is to identify frequently-used substructures in good individuals by evolutionary or other means, and encapsulate them as intermediate building blocks.

An example of ADF use appears in Fig. 4 [47]. GP trees in an ADF system have two functional parts: branches that produce results, and at least one function defining branch. A normal GP tree evolves in the result-producing branches, while the branches in the other part evolve named subtrees as functions. These function definitions are treated as terminals in the result producing branch, and can be called many times in constructing an individual. ADFs share the above three characteristics, connectivity, relocatability, and re-use, with most modularising systems. All have achieved substantial improvements in GP search [48].

5.1.3 Modularity in PMB-GP

Modularity is not directly supported in standard GP, though it is implicit in the operation of crossover; explicitly supporting it requires extensions such as ADFs, and this explicit support provides benefits in terms of search speed.

A similar situation obtains in PMB-GP: the basic models do not support modularity directly. It would be complex (but not impossible) to directly incorporate explicit support for modularity into PMBA individual representations and into models. For example, there is no intrinsic problem in encoding ADF representation into a PMB-GP. But the increased complexity would inhibit analysis of PMB-GP; since ease of analysis is one of the principal arguments for PMBA methods, this may be undesirable. At any rate, to date most PMB-GP systems have taken indirect routes to supporting modularity, by providing support for specific aspects.

In the remainder of this section, we examine two critical factors in designing PMB-GP models: local dependence and positional determinacy. Local dependence is an important factor in connectivity, and positional determinacy is closely related to modularity, and especially to relocatability. We explain these concepts and then illustrate how PMB-GP model construction can address them.

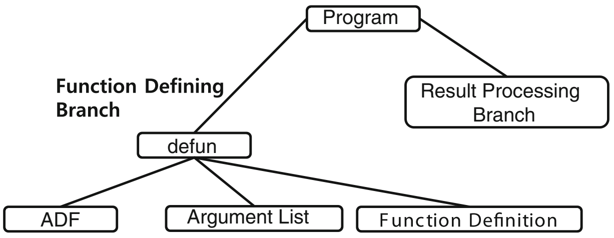

Fig. 4 Example use of automatically defined functions in GP

5.2 Modularity and locality of dependence

In GP, the semantics of an individual are defined through the interaction of three components:

1. the structural constraints
2. the evaluation mechanism for the structure built under those constraints
3. the fitness function that maps the result of the evaluation to a fitness value (or more generally, to a fitness ordering)

The evaluation of individuals is almost always defined recursively over the structural constraints: for example, the evaluation of $+$ is defined recursively on the evaluation of its two arguments. But the application of evolutionary or PMBA methods to a search domain implies a further assumption: that there is some correlation between fitness and the evaluation of individuals (if there were no such correlation, there would be nothing for these search methods to take advantage of, and we might as well use random search). Combining these two perspectives, we have prima facie reason to expect that this might induce dependence between the desirable values for adjacent nodes (and in particular, parent-child node pairs). Returning to our example, suppose $x + (-1)$ is a highly fit component. We would anticipate that, if we replaced $+$ by $-$, it would also be necessary to replace -1 by 1 to retain that high fitness (i.e. $x - 1$). In evolutionary GP, we see this local dependence embodied in the shape of building blocks.

This has implications for the properties of PMB-GP model representations: it is desirable for them to be able to directly represent such local dependences. Beyond the issue of detecting problem-specific dependences, PMB-GP models have different ability to represent such dependence. It leads to varied ability in detecting complex and sophisticated dependences, no matther what the problems are. Current implementations vary in the methods they use: some represent all such dependences in the model structures [13, 14, 123], while others use an explicit random variable to store the dependence [1, 31–34, 42, 72, 89, 93, 97, 98, 104–106, 114, 129–131]. It is by no means clear, at this point, what is the most appropriate mechanism to store such dependences.

For example, in the GACP and bACP systems explained in Sect. 6.3, trees are represented by linear strings in prefix notation [12, 93]. Learning and sampling can be done as in any other PMB-GP system, so there is no problem in building a system in this way.

Nevertheless, the structure of any building block learnt by these systems differs substantially from what would be learnt in a direct tree representation. For example, the downstream nodes of a particular node may contain a mix of sibling and child nodes from the equivalent tree, so that what might subsequently be learnt as a building block in the model is a complex mix of sibling and child semantics. Thus the implicit assumptions about local dependence are different from those that underlie other representations. In grammar systems, for example, any captured local dependence will be maintained as a tree form, but in ant-systems, while there may be a tree form, it is combined with extra connectivity to sibling components.

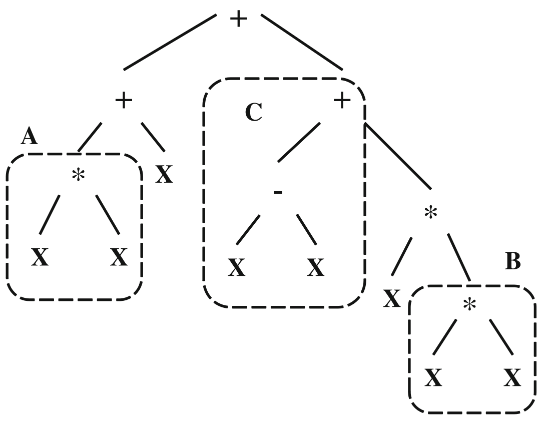

Fig. 5 Illustrative Example: Position, Building Blocks and Introns

5.3 Modularity and relocatability

Many PMB-GP systems use a position-determined (PD) model,[2] as in most PMB-GAs: the system attempts to learn what symbols should be in which specific position of a PPT [98]. However as we noted earlier (Sect. 5.1.1), this conflicts with one of the underlying assumptions of tree-based GP, the relocatability of building blocks (as reflected in the importance of crossover in GP). But this assumption might be mistaken: perhaps the relocatability of building blocks in tree-based GP is an accidental effect of typical subtree crossover, rather than an important property.

Consider a GP or PMB-GP system applied to a well-known symbolic regression problem, in which the target is a function fitting points from the function:

$$f(x) = x^3 + x^2 + x.$$

Figure 5 shows an example of a typical individual I_1 that might appear in a GP or PMB-GP population during such a run. In fact, since it is a perfect solution, it is likely to appear in the selected population as well. Suppose that our algorithm is still running (for example, imagine that the data is noisy, so that we do not at that point know that we have found a solution). What would we like a PMB-GP system to learn from I_1?

We note that block C is an intron—it has no effect on the semantics of the individual; an individual I_2, consisting of I_1 with block C omitted, will have exactly the same semantics as I_1, and hence exactly the same fitness. In learning a non-PD model, the two copies of block C in individuals I_1 and I_2 may reinforce each other. On the other hand, in a PD model, they will not only not reinforce, but actively conflict with, each other. Thus introns may actively inhibit learning in PD models. This may be undesirable, though it may also have desirable effects: we can expect

[2] By ‘positional determinacy’, we mean that in generating, or learning from, an individual, the decision over which random variable to use is completely determined by the absolute location of the specific node relative to the root. This concept has been previously used by Shan et al. [107] using the term ‘positional dependence’. Unfortunately their terminology has sometimes led to confusion: particularly in a statistical context, ‘positional dependence’ covers cases where the dependence is probabilistic, rather than deterministic. We use ‘determinate’ to avoid any ambiguity.

more rapid learning in individuals without introns, thus over time introns may be eliminated from the models.

5.4 A framework for evaluating modularity in PMB-GP systems

In this subsection, we introduce a framework for evaluating modularity in PMB-GP systems. The key difference between classical PMB-GAs and PMB-GPs, from a model-learning perspective, is the variation of the size of individuals in a population—i.e. of the number of observable locations. But at least within a single generation, the model size is fixed. Thus the mapping between model variables and observables is generally not a bijection. Different PMB-GP systems take different approaches to mapping from the statistical model's random variables to the observable locations. We use this mapping as the basis of our framework. To clarify these concepts, we first carefully discuss the learning environment that arises in PMB-GP. We then explain how this leads us to a framework enabling us to distinguish different kinds of PMB-GP. Finally, we examine how this allows us to better understand modularity in PMB-GP.

5.4.1 The learning environment of PMB-GP

To clarify the learning environment, we need to make a distinction between the space in which the search is conducted (which varies from system to system), and that in which individuals are evaluated for fitness (for PMB-GP, in most cases, a space of expression trees). This corresponds closely to the distinction drawn in GA between genotype and phenotype spaces. In a slight abuse of terminology, we use the same phrasing here: the genotype space is the space from which the probabilistic model is directly learnt, and in which the next generation's samples are drawn, while the phenotype space is (as in genetic systems) the space in which they are evaluated. We do not mean to imply that this learning/sampling process is a genetic process, though it has interesting analogies and can be viewed as a generalisation of mutation and crossover operators.[3] As in genetic systems, the genotype and phenotype spaces may be identical (as in PPT-based systems) or distinct (as in grammar-based systems).

Phenotype Space The phenotype space for PMB-GP consists of all syntactically correct individuals. In the search process, a small subset of this space is observed, and used to learn the stochastic model. In GP, it is often a space of expression trees, in which an individual is a tree composed of nodes labeled by an appropriate symbol set. For example, the labels for symbolic regression trees comprise arithmetic operators, constants, and variables.

Genotype Space For the same reasons as in genetics-based evolutionary systems, the phenotype space may not be the most suitable space in which to conduct search. Many PMB-GP systems do not directly learn their probability model from the

[3] From another perspective, a typical GA can be modelled as a PMBA in which crossover and mutation act to change the distribution over the phenotype space [90].

phenotypes, but instead maintain a separate representation over which model learning and sampling are performed.

For example, in grammar based systems, learning is conducted on the space of derivation trees from which the expression trees are generated. Derivation trees are mapped to expression trees by a many-one mapping. In linear GP systems, the genotype space is a linear string, though the phenotype space may consist of expression trees. In PPT-based systems, the genotype space is the same as the phenotype space—i.e. expression trees.

The genotype representation is important in PMB-GP systems, since it directly affects the ease with which we may adapt classical PMB-GA techniques. When the genotype is linear, the similarity to GA genotypes means that we can readily adapt PMB-GA techniques, no new theory is needed. We refer to such systems as GA-like. When the genotype is tree-structured, and we pay attention to this tree structure in learning the statistical model, new techniques are needed; we refer to such systems as GP-like.

Modularity and Genotype Space Ultimately, what we desire is modularity in the phenotype space, since that is what governs the semantics of our solutions. However sampling from the probabilistic model generates elements of the genotype space. For example, when we use stochastic CFGs as the statistical model, the directly-sampled derivation trees retain a record of the sampling process, and directly retain information about how the CFG productions generated specific components of the tree (and thus created modular structures), whereas the genotype space (expression trees) do not. Modularity in the phenotype space is an indirect by-product of the application of the genotype-phenotype mapping to modular structures in the genotype space. So we will refine our focus to the representation of modularity in genotype space.

There may be prior knowledge about likely building block structures in specific domains (and thus representations which are able to represent such knowledge may be useful). However if we do not have such knowledge, then our system needs to be able to discover (and therefore represent) it.

Sampling and Learning In any statistical learning system, we need to pay attention to two key aspects:

- How samples are created
- How updated samples are made available to the model learning system

Both these issues are relatively straightforward in PMBAs, so they are little discussed. However in PMB-GP they are both more complex and more varied. But we can nevertheless assume:

- Sampling proceeds generatively, either sequentially or recursively, over a tree structure
- All nodes of the genotype are available to the learning system (i.e. the observables are all nodes in the genotype)

In particular, in grammar-based systems, the genotypes (derivation trees) contain nonterminals that do not appear in the final phenotype; nevertheless, from the perspective of the learning system, they are observables.

5.4.2 Context and position

The generative sampling of PMB-GP individuals induces a partial ordering on the constituent nodes. This ordering determines which nodes are required to be already sampled before another is sampled. Thus we can, at least for all current PMB-GP systems, define an absolute position for a node, in terms of the construction path from the starting node. In turn, this leads to the notion of positional determinacy (PD), which we will explore further below. In most discussions of PMB-GP, this absolute position is contrasted with the context of a vertex—the contents of its surrounding vertices—and the corresponding notion of context determinacy or dependence (CD).

In the most generalised models, probabilistic reasoning is based on the full joint distribution between all variables of the model. However in most realistically-sized applications of statistics, including PMB-GP, the full joint distribution is useful only as a theoretical concept. Using it directly is, in general, infeasible both statistically (we could never generate enough samples to accurately approximate the distribution) and computationally (we could not store the distribution, and even if we could, we would run out of time to compute with it). In EDA systems there is a further issue: we may not even want the full joint distribution even if we could practically obtain it. Generalisation resulting from approximations to the distribution may lead to more effective search. For all these reasons, we typically look for factorisations of the distribution that correspondingly allow us to consider smaller subsets of the variables in relative isolation.

In GP of any kind, values for each variable cannot be directly observed (i.e. correlated with fitness) in isolation. They are only observable within an individual—i.e. together with values for all other variables. So finding a factorisation involves finding a subset of the variables whose values are sufficient to explain the value of the variable under consideration. To simplify terminology, for a given vertex v, a context C consists of sets of values for any subset of the remaining vertices, other than v, indexed by the values of v. In particular, in the full context, all variables other than v are included. Based on the partial ordering of nodes induced by the structural definition, it is useful to distinguish between four other kinds of contexts:

1. Ancestor Contexts: all vertices are ancestors of v
2. Descendant Contexts: all vertices are descendants of v
3. Incomparable Contexts: no vertex is either an ancestor or descendant of v
4. Mixed Contexts: a mixture of ancestor, descendant and incomparable vertices

These distinctions will be useful for clarifying the relationships between different kinds of PMB-GP models, as we will see below: the distinctions between many of the different systems can be seen as primarily due to differences in the contexts that they rely on for learning.

5.4.3 Strategies for finding and constructing building blocks

Detecting Relocatable Building Blocks Learning a statistical model from the selected population is a process of storing information about the population in the random variables of the model. In doing so, the first issue is how to detect the locations of potential building blocks that might correspond to a specific random variable. In PMB-GP, since the size of individuals is flexible, we may have incomplete or even no knowledge of where a specific value or set of values might be observed. Since we can search through a set of candidates, the issue is not so much how to detect building blocks, as how to determine the candidate locations to test.

In defining such locations, we face more complex problems not easily handled by models assuming positional meaning (where the problem is simply to estimate the appropriate outcomes at known locations). The flexible size of GP individuals leads naturally to representations in which one random variable might map to many locations, while the structural constraints on GP trees imply specific connections between components. In practical PMB-GPs, this is handled in a variety of ways. In PIPE, for example, we map the observable location of the leftmost child of the root to a specific random variable, the next but leftmost to another and so on (i.e. relying on the absolute position). By contrast, variables in GMPE are mapped to locations where the parent symbols and labels of variables match (i.e. relying on the context). In ant-based systems, or similar systems with transformed tree individuals, it is more complicated to specify the mapping rules. There are many different rules and strategies, with little standardisation so far.

Constructing Structured Building Blocks Even once we have determined the mapping between random variables and locations, we still have to decide on a structure to represent building blocks. Building blocks are generally regarded as structured, connected subgraphs of relationships between variables. This raises issues of locality of dependence. Depending on the assumptions about local dependence between subcomponents, different variable construction mechanisms will be needed to capture these dependences.

We can illustrate this more concretely by considering three grammar-based systems of increasing complexity. At one extreme, scalar SG-GP (sSG-GP [89]) cannot create structured building blocks. It can only generate building blocks that happen to be representable in the specific problem grammar. At the other extreme, GMPE, which can learn a new grammar as its search progresses, can in principle represent tree segments of almost any given complexity. Between these extremes lies ACO-GP, with a prefix-notation linear representation. The prefix notation can in principle again represent arbitrary tree segments, but the representation of non-left-connected components requires discontinuous linear segments, which may be difficult to detect.

5.4.4 Mapping to observable locations: position versus context

Position-based mappings such as in PIPE are the simplest form. In all systems we know of, they are one-to-one (though they may be partial): each random variable maps to at most one genotypic location. They allow for variation in genotype shape and size by failing to map some variables. As a consequence, the model size is at least equal to the largest individual size—leading either to very large models (with consequent requirements for very large sample sizes) or to fairly strong restrictions on individual size.

In pure context-based mappings, such as sSG-GP or GMPE, the mapping ignores positional information and relies solely on context (almost always, an ancestral context). Given a specific context, whatever its location, a specific random variable will be used. The mapping may be fixed (as in SG-GP), or may change as the model is learnt (as in GMPE).

We also see mixed mappings, primarily context-based, but also incorporating some positional information (as in the use of depth in vSG-GP and Program Evolution with Explicit Learning (PEEL) [105]).

5.5 Capturing modular building blocks

Having introduced some basic terminology and concepts, we now discuss in more detail some of the issues involved in capturing building blocks

5.5.1 Positional determinacy

We previously pointed up one of the issues involved in positional determinacy: that PD models are unable in principle to represent (and thus to learn) relocatable building blocks. In such systems, building blocks can only be learnt in fixed positions. We contrasted this with context-based models such as those supported by grammar-based systems. These can readily represent relocatable building blocks, but generally will have great difficulty in learning positional building blocks (if they can represent them at all). Of course, it is important to note that these two extremes are not the only possibilities. For example, vSG-GP and PEEL both use depth as an annotation in stochastic CFGs (always in vSG-GP, discretionary in PEEL). Thus they are partially determined by absolute context, with corresponding restrictions on the building blocks they may learn, so that vSG-GP can learn building blocks that are relocatable across positions within a given depth, but cannot share these building blocks between different depths.

A similar situation arises with context-based learning. Perhaps the same structured component may be found in two different contexts. Systems based on fixed-context models (e.g. most grammar- based systems) will not be able to share learning between the contexts, so that the two components must be learnt separately. Thus it may be necessary to be able to structure new contexts, as provided by the annotation learning of PAGE, or the grammar learning of GMPE.

5.5.2 Detectability of building blocks

Given the above, it might seem that the problem is merely that in position-determinate systems there are too many model variables, so that modular structures are split among them, compromising learning. Thus a potential solution is to simply reduce the number of model variables, sharing many locations among each. The extreme is systems such as sSG-GP, in which model variables correspond to the (usually small number of) grammar nonterminals. However here we see the opposite problem: that each variable shares so many genotype locations (all that derive from the same nonterminal) that multiple building blocks are liable to overlap and thus contaminate the learning of each other, leading to the problem of detectability.

The tension between these two aspects, building block relocatability requiring multiply-mapped model variables, but detectability leading in the opposite direction, is what has led to some of the more recent developments, of systems which can gradually learn new mappings, and thus avoid some of the conflict.

6 Probabilistic model-building GP systems

This section provides a survey introduction to the wide variety of PMB-GP systems in the literature, in a discussion based particularly on some of the more distinguishing of the issues raised in Sect. 5.

ACO-GP systems have been developed almost entirely separately from EDA-GPs (in the two literatures, one finds few cross-citations, even when they are discussing the same issues). In fact, one of our aims is to point out the similarities between the two fields, and to encourage greater cross-fertilisation. Hence we have grouped ACO and EDA-based systems together, using the genotype structure as the primary distinction, and the model structure as a secondary criterion.

Among PMB-GP systems, we may distinguish between those that use GA-like (i.e. linear) genotype representations (naturally leading to probability models resembling those of conventional PMB-GAs), those that use conventional-GP-like (i.e. tree-based) genotype representations, and those that use more general directed graph representations. The latter two often require new probability models, more distantly related to those of the PMB-GAs. This leads us to the following primary categorisation:

6.1 Systems with tree-GP-like representations
6.2 Systems with more general directed graph representations
6.3 Systems with GA-like representations

At the top level of the categorisation, we use the form of a representation, rather than its semantics, because that is what is available to the model-learning process. For example, a number of systems use linearised representations of trees (e.g. prefix representation). Unlike direct tree representations, in these systems, the tree information is not available to the model learning system (for example, in a direct tree representation, the information about which node is the tree ancestor of a given node is directly available to the model-learning algorithm—whether it is used or not—whereas in a prefix notation, it is generally not available).

Within the top-level categorisation, we further subdivide into different uses of these representations (for example, within tree-GP-like representations, we distinguish between representations which directly represent the expression tree, and those which use a syntax tree as genotype).

Within these levels of categorisation, we may further distinguish systems, based on the complexity of the model and the learning process. The simplest systems use simple univariate models. The next stage of complexity uses fixed multivariate models (for example, bivariate dependencies between tree children and parents), while the most sophisticated systems learn the model structure itself. However the number of systems within each such sub-category is small, so this level of categorisation is only represented informally (within each second-level category, the systems are arranged roughly in order of the level of sophistication of the probability model).

Table 2 summarises the overall categorisation of the systems we consider here.

The order of presentation of systems is not intended to be significant (and in particular, is not temporally based). Instead, the arrangement has aimed at economy of presentation. Since later systems often build upon the ideas of earlier, this inevitably leads to some degree of temporal order. But this is a by-product of the research process, not an intended result. In particular, the fact that a particular system occurs late in the presentation is not intended to reflect on either its research priority or its importance.

In discussing these systems, we relate them as far as possible to the issues raised in Sect. 5. Specifically, we aim to cover their structural formalism (in terms of AND/OR graphs) and semantic issues (positional determinacy, context dependence, and detectability). The degree to which we will be able to do this varies from system to system. Some are described in full detail in the literature, and hence can be fairly unambiguously classified; for others, the available descriptions are relatively brief, so that classification turns on the exact meaning of one or two words; we hope that, in discussing these systems, we have done full justice to them.

Where possible, we have used the authors' own names and abbreviations to describe their systems. However this has not been possible in all cases, either because the authors do not name their systems, or because the names are too susceptible to confusion with other systems. In these cases, we have introduced names and abbreviations that we hope are sufficiently descriptive to avoid confusion.

Finally, we review a small number of systems that can be viewed as hybrids of PMB and other GP methods.

6.1 Systems based on GP-like representations

In this group, there are two main kinds of stochastic models: PPT models and stochastic grammars such as SCFG. They are clearly distinguishable, so we detail them separately. In both cases, the development has proceeded in a relatively orderly way, with increasing sophistication of the probability models, so that they are presented in roughly chronological order. We conclude with a small number of hybrid systems not fitting into either category.

6.1.1 PPT based systems

The detail of the PPT structure and an example of how it works were provided in Sect. 3. The mapping of observables onto random variables is based on matching of positions, so it is a position-determinate model. This property is shared among all the PPT-based systems. The major variation between these systems lies in the

Table 2 Summary: PMB-GP models and their determinacy/dependence properties

Representation			Models	Positional		Context (Label)		Context (Ancestral)	
Genotype		Model		Det.	Dep.	Det.	Dep.	Det.	Dep.
Tree GP like	Expression Tree	PPT	PIPE	1	▨	-1	-1	-1	-1
			EDP	1	▨	-1	-1	-1	1
			ECGP	1	▨	-1	-1	-1	1
			POLE	1	▨	-1	-1	-1	1
			AP	1	▨	-1	-1	-1	-1
		N-gram	OFGP	-1	-1	1	▨	-1	1
	Derivation Tree	Stochastic Chomsky Grammar	sSG-GP	-1	-1	1	▨	-1	-1
			vSG-GP	-1	1	1	▨	-1	1
			PEEL	-1	1	1	▨	(-1,1)	1
			GMPE	-1	(-1,1)	1	▨	(-1,1)	1
			PAGE	-1	(-1,1)	1	▨	(-1,1)	1
			CFGT	-1	(-1,1)	1	▨	(-1,1)	1
			CSGR	-1	(0,1)	1	▨	-1	1
Graph			GNP-EDA	▨	▨	▨	▨	▨	▨
GA like	Prefix Exp. Tree		bACP	▨	▨	▨	▨	▨	▨
			gACP	-1	▨	-1	-1	-1	▨
			GACP	1	▨	-1	-1	-1	1
			DAP	-1	(0,1)	1	▨	(-1,1)	1
	Prefix Der. Tree		GAP	-1	-1	1	▨	1	▨
			EGAP	-1	-1	1	▨	1	▨
			GBAP	-1	-1	1	▨	1	▨
	Linear Genotype		BAP	1	▨	-1	-1	-1	1
			BOAP	1	▨	-1	-1	-1	1
			CGP-EDA	1	▨	-1	-1	-1	-1
			N-gramGP	-1	-1	1	▨	-1	1
			AntTAG	-1	-1	1	▨	-1	-1
Hybrid			CFGR	▨	▨	▨	▨	▨	▨
			MOSES	▨	▨	▨	▨	▨	▨
			PAM-DGP	▨	▨	▨	▨	▨	▨

Abbreviations Determinate (Det.), Dependent (Dep.), Expression (Exp.), Derivation (Der.)

Entries : For each entry, the numerical value indicates whether the corresponding property (determinacy or dependence) holds in the corresponding form (positional, label context or ancestral context). A value of 1 indicates that the property holds; -1 indicates that it does not hold. 0 indicates an implicit rather than direct dependence. If the property can change during evolution, we indicate this by a range (-1,1). A greyed-out region indicates that the corresponding property is inapplicable or (in the case of dependence) that it is implied by determinacy.

sophistication with which they can represent probabilistic linkages between random variables, and what is closely related, the detectability of specific structures.

Population-based Incremental Program Evolution (PIPE) was the first PMB-GP system, and introduced the PPT representation [98]. Its detailed processes were presented as an example in Sect. 3. Although it is described in EDA-like terms (and is usually classified so in discussion of PMB-GP), it is worth noting that its probability update does not attempt to estimate the posterior distribution accurately, but simply moves the model distribution in the direction of the sample distribution;

in this respect, it resembles typical ACO-GP systems rather more than EDA-GP. It uses independent variables and a fixed mapping of observables onto variables, and thus has a minimum of detectability in comparison with other PPT-based systems. The use of an independent variable model is unsurprising—PIPE was first described only three years after PBIL—but limits PIPE's ability to find complex programs. Subsequent extensions mirrored those in EDA—the simple independent model of PIPE (PBIL) was subsequently extended to represent more dependences between random variables.

Ant Programming (AP) is the first example of an ACO-GP [96] explicitly described as such. The genotype and model structure seem to us identical to PPT; the learning mechanism is also very similar, though explicitly described in terms of pheromone tables and evaporation (the exact update formulae differ from PIPE, but the similarities are more obvious—once the terminologies are equilibrated—than the differences). As a result, the considerations regarding positional determinacy, detectability etc. are as for PIPE.

Estimation of Distribution Programming (EDP) extended PIPE by providing a more complex PPT-based statistical model, in which a variable mapped to a specific node may be dependent on the variable mapped to the parent of that node [129–131]. This reflects a common intuition, that the distribution of child nodes may be affected by the choice of parent nodes. as compared to PIPE, it expands detectability, representing a more sophisticated distribution describing the same observables.

Extended Compact Genetic Programming (ECGP) further extended the complexity of the probability model over the basic PPT structure. It captures multivariate dependences [104]. It takes its mechanism for learning a conditional distribution from the Extended Compact Genetic Algorithm(ECGA) [30], using marginal product models (MPM). The random variables are partitioned into discrete groups; within each group, variables are fully dependent; between groups, the system assumes independence. Thus the key issue is how to partition the variables. The search for a suitable partition uses the minimum description length (MDL [92]) principle to determine preferences between models. Sampling relies on this distribution as in ECGA, but has also to satisfy the constraints imposed by the PPT structure. In comparison with PIPE or EDP, ECGP can represent more detail of the distribution through applying more conditions to each variable. The most important difference from PIPE and EDP lies in the learning process. PIPE and EDP rely on a single fixed model structure, which does not change over time. All learning is performed through updating probabilities alone. However ECGP, by learning a partitioning, can change the structure of its model over time.

Program Optimisation with Linkage Estimation (POLE) aimed to provide a more flexible model of interdependence [31, 33]. In this respect there is a close parallelism between PMB-GA and PMB-GP systems. In the former, ECGA learns better factorisations of models by creating MPMs, but ignores all interdependence between the MPMs: it requires either full dependence or full independence. The Bayesian Optimisation Algorithm(BOA) [83] overcomes this restriction by learning a Bayesian network interlinking the variables. In the latter, POLE bears a somewhat similar relationship to ECGP. The extension from the GA to GP world is similar in both cases, but BOA and POLE differ in their approach to sampling, because generating a GP tree individual must still respect the PPT structure. At each iteration, POLE learns a Bayesian network superimposed on the PPT representation, estimating the internal linkage from observed information (i.e. the selected population). The next population is generated following both the structural constraints of the PPT and the probability distribution of the current Bayesian network.

6.1.2 Other expression tree representations

Operator Free Genetic Programming (OFGP) [35] uses the expression tree as genotype, and learns from it using an N-gram ancestor context model. N-gram models are mainly familiar from document processing, but can also be used to describe ancestor context in trees. Such models store the frequency histogram of N-Gram components, and use them to sample new (hopefully fit) individuals. OFGP ignores absolute position, so that it is not position-determinate; however it is fully ancestor-context-determinate up to the chosen size of the ancestor context.

6.1.3 Systems based on stochastic Chomsky grammars

Many grammar-based models use SCFGs, and others use mildly context-sensitive grammars, either modified Chomsky-style grammars, or in some cases, a non-Chomsky grammar form. We will detail the specific form of grammar as we discuss each system. Here, we collect together all the systems which use Chomsky grammars, since they relate to each other in easily understood ways; subsequently, we will look at systems using non-Chomsky grammars. These Chomsky grammar models are position-indeterminate in principle, because the mapping from observables to variables relies on matching symbols, rather than positional information. This does not mean that the mapping is always position-independent, as noted in Sect. 5.4: depending on the specific grammar, it is possible for the grammar to constrain the mapping to any degree of position dependence (for example, it is possible—though probably pointless—to write SCFGs exactly equivalent to PPTs). In grammar-learning systems, it is common for there to be a gradual increase in position dependence as the system converges toward a specific solution. As with PPT models, detectability is a key issue that has been emphasised in the development of the field; it is typically improved in this case, either by learning more complex grammars, or by learning supra-grammatical structures that can represent more complex distributions.

Scalar Stochastic Grammar-based Genetic Programming (sSG-GP), the first grammar-based PMB-GP, was introduced in Sect. 3 as an example of the use of SCFG [89]. The original paper introduced two versions of SG-GP: scalar SG-GP(sSG-GP) (the system described in Sect. 3), and vectorial SG-GP(vSG-GP), an extension incorporating some positional information (see below). sSG-GP is fully context-determinate: the mapping of a specific observable is determined only by the nonterminal (itself determined by context) labelling the node. As a result, sSG-GP not only can, but must, share information about the same building block occurring at different depths. This limits the complexity of the distributions that can be represented—and consequently, of the problems that can be solved.

Vectorial Stochastic Grammar-based Genetic Programming (vSG-GP) was defined in the same paper, in an attempt to overcome the limitations of sSG-GP [89] by incorporating some positional (depth) information. Assuming a depth limit D, for each sSG-GP variable, vSG-GP has D variables, one for each depth. An observable with a given label is mapped to the variable with the same label and depth. Thus the vSG-GP model is partly determined by position and partly by context. vSG-GP thus supports a more complex overall distribution, and can solve problems whose solution cannot even be represented by sSG-GP models. On the other hand, vSG-GP cannot share information about the same building block occurring at different depths. This is an example of the previously-noted trade-off between context-determinacy and detectability. Much of the subsequent work can be viewed as an attempt to overcome this dilemma, by avoiding fixed probability models (as in the two versions of SG-GP) and instead learning structures on-line, aiming to reflect the actual dependencies in the problem domain.

Program Evolution with Explicit Learning (PEEL) uses an extension of the vSG-GP model [105]. As in vSG-GP, PEEL can use the depth information of variables. However the more important difference from vSG-GP is that the use of the depth information is not compulsory. Rather, PEEL starts off with a model like that of sSG-GP. However once the distribution of a nonterminal satisfies a specific condition, the corresponding random variable is split into a separate variable incorporating the index information. Thus PEEL is position-indeterminate, but can become context-determinate. In the extreme case, it is possible that each random variable represents the distribution of an observable located at one specific ancestral context (for combinatorial reasons, this would be highly unlikely to arise in all but the tiniest problems). Splitting variables allows PEEL to fairly search among unknown ancestor contexts. Building blocks may be shared across different depths, or limited as the system learns specific depths. Building blocks are always shared horizontally across locations, so that PEEL (like vSG-GP) is unable to specialise its model horizontally.

Grammar Model-based Program Evolution (GMPE) [106] aimed to overcome these limitations of PEEL, by learning not just specific locational information, but a full CFG to incorporate previously-learnt restrictions. The underlying idea of GMPE is to specialise the grammar describing the original search space to a new, more restricted, grammar covering only the desirable parts of the search space. Thus the grammar restrictions take the place of the depth limitations of vSG-GP and PEEL. GMPE's fundamental task is, given a selected sample of individuals and a prior grammar, to find a sub-grammar of that prior grammar that describes the selected sample well. This is essentially a machine learning task. As such, it could be undertaken in either specialising form (starting from the prior grammar and restricting it to more specifically cover the selected individuals), or in generalising form (starting from the extreme specific grammar, covering only the selected individuals, then generalising it toward the prior grammar). GMPE takes the latter approach. Doing so requires that the selected individuals generate an ordering over grammars (so that a heuristic search may be conducted). GMPE uses informationtheoretic measures (Minimum Message Length—MML—[119]) to direct this search, which is conducted using a ’merge’ operator that unifies two non-terminals (i.e. random variables) and their corresponding distributions. In its original form, GMPE uses the initial grammar as the prior grammar in all generations (so that learning is not fully incremental).

Merging of symbols in GMPE changes the observables mapped to a variable, with no restriction other than that imposed by the initial grammar, while new variables in PEEL must share ancestral contexts. This aspect is an important issue, because the restriction of PEEL on mapping observables can cause implicit bias when we aim to search for a better mapping without any prior knowledge. Detectability can be handled by structure learning, without artificial limitations imposed by previous groupings of observables.

Context Free Grammar Transformation (CFGT) [13], Bosman and de Jong’s previously unnamed system, generates individuals from an SCFG, as in the preceding systems. However it differs from these systems in an essential way, in that genotypes are represented purely as expression trees, not as derivation trees. Thus when it comes time to learn a new grammar from the selected population, the individuals must first be parsed. Since multiple parses could lead to complexity, grammar ambiguity is an important issue for this system. Nevertheless in many ways, it can be seen as a close analogue of GMPE, the major difference being that it learns directly from the expression trees rather than from GMPE’s intermediate genotype, the derivation trees.

Program with Annotated Grammar Estimation (PAGE) resembles GMPE in that segmentation of observables depends only on symbols. However unlike GMPE, these symbols are not CFG symbols, but rather annotating symbols [32, 34]. The idea is inherited from probabilistic context free grammars with latent annotations (PCFG-LA) [58] in computational linguistics. As in GMPE, there is an initial CFG, representing the whole solution space. The nonterminals in observed individuals carry additional arbitrary annotations, (bounded in number by a predefined limit, the annotation size). Random variables thus correspond to a combination of nonterminal symbol and label. To learn a suitable distribution for the variables, PAGE uses the EM algorithm [20] or variational Bayes [9]. In the annotation process, the EM algorithm can generate annotations for a symbol without restriction. Detectability and context determinacy are quite similar to GMPE: in the extreme, both can learn completely specific trees, and the only restriction on their learning relaxations of this extreme is symbol matching (and in the case of PAGE, the annotation size). However a key difference lies in their mechanism for choosing between model structures (MML vs EM).

6.1.4 Other grammar representations

Context Sensitive Grammar Refinements (CSGR) is our proposed name for Tanev’s unnamed context sensitive grammar (CSG) system [114]. Tanev extended SCFG to stochastic CSG (SCSG). In an SCSG, the left-hand-side nonterminal contexts are the preceding terminal sequences of tree individuals, rather than parts of the ancestor contexts. The contexts of observables mapped to a single variable are more specialised than in SCFG, being restricted by the preceding terminal sequence. As with the other grammar-based models, CSGR is position-indeterminate, but context-determinate. It can represent a more complex variable-observables mapping in comparison with SCFG, however it is limited in relocatability since it is restricted from noting building blocks across different preceding terminal contexts.

6.2 Systems based on general directed graph representations

As in genetics-based GP, it is possible to use directed graphs rather than trees as the genotype (and phenotype) representation. One such system is GNP-EDA.

Genetic Network Programming with Estimation of Distribution Algorithms (GNP-EDA) uses directed graph representation for individuals [52, 53]. The probability model is a matrix $P(i, k, j)$ giving the probability of a connection from the k^{th} branch of node i to node j (it is not completely clear whether this is initialised with uniform or random probabilities). Network graphs are sampled from this model, evaluated and selected, as in other PMB-GP. The probabilities of particular links are updated to reflect the selected population, by an update more reminiscent of ACO than EDAs. The system is sufficiently different from other PMB-GPs that it is difficult to directly compare. In particular, there is no clear meaning to position, unlike expression tree or string representations. For example, the start node is clearly the analogue of the root node of a tree-based GP. But the same start node may also be the third node in a three-cycle from itself, thus also having a position at depth 3. Thus it is difficult to define positional determinacy. Similarly, there is only one node label ("judgment node"), hence there is no useful meaning to be assigned to context.

6.3 Systems based on GA-like representations

In GA-like representation systems, both in genetically-based GP and in PMB-GP, the genotypes of individuals are represented by linear chromosomes. There are a number of ways to achieve this.

Some systems, inspired by the fact that computer programs are themselves linear structures, directly use a linear representation. This has a number of advantages in terms of direct transfer of technologies and tools from GA systems. But it also has a downside, in that components that would be recognised as homologous by a human observer may lie in different positions, and hence may not have any homology from a genetic or PMB perspective.

One way of overcoming this is to constrain the representation in some way, via structural constraints. In a genetic system, operators (especially crossover) may be restricted, so that the genetic operators may only be applied in restricted ways. Analogously, in PMB systems, sampling and learning phases may be constrained in ways that increase homology between different individuals. This may increase the effectiveness of the operators (by ensuring that they act homologously), but at the cost of reducing the similarity to GA, and hence affecting the transfer of GA technologies.

An alternative is to treat the linear genotype as a recipe for constructing a structured object (usually, a tree). Thus the genotype itself is linear, but prior to evaluation, is translated to a structured phenotype, which is the object that is evaluated. In this case, the issue of homology still arises: the level of homology will depend both on any restrictions that may be imposed on the evolutionary stages (genetic operators in the case of genetic systems, sampling and learning in the case of PMB), and on the extent to which the translation process preserves homologies.

Overall, this leads us to a threefold sub-classification of such linear representations:

1. Prefix notation (implicit or explicit) for expression trees
2. Prefix notation (implicit or explicit) for grammar derivation trees
3. Other linear representations

In all cases, extension of standard PMB methods still face a key hurdle: it is a hallmark of GP problems that the solution complexity is not predefined. There may be a complexity limit—such as a tree size or depth limit or a string length limit—but it is only an upper bound, the system is free to construct smaller solutions, and generally this is desired. Conversely, stochastic models designed for classical optimisation problems are mostly of fixed complexity—whether for discrete or continuous domains. The classic solution in PMB-GP is to define a fixed-complexity genotype, of sufficient size to have reasonable certainty that it can represent a solution, and to permit part of the genotype to represent "undefined" values. As a result, most models used with GA-like representations are position-determinate. However they vary greatly in syntactic style and detectability, as learning more sophisticated structures to represent more complex distributions has been an important research direction.

6.3.1 Prefix notation expression tree genotypes

Ant Colony Programming (ACP) causes some confusion, because the same name has been used for two rather different systems. To distinguish them, we prepend the first author's name, as below.

Boryczka Ant Colony Programming (bACP) [10, 11] actually introduces two different genotype representations—prefix-notation expression trees, and sequences of assignment statements (we were unable to determine from the description how the system deals with incomplete sequences of assignments which might lead to undefined values). These two representations are combined with two different sampling methods, a purely stochastic one in which any symbol may be sampled any number of times, and a second in which each symbol is replicated a fixed number of times, but each replicate may be used only once. In the second (tabu list) form, it appears that multiple instances of the same symbol have the same probability, although this is not completely clearly specified. The probability model is a classic parent-child dependency model. In the assignment statement representation, neither genotype position nor context have a clear meaning, so it is not possible to assess related properties of the representation. In the expression tree representation, as a consequence of the use of prefix notation, the mapping of observables onto variables is not determined by the position in the phenotype expression tree (if the arity of a variable in an ancestral context changes, then the location corresponding to the current variable may change). Detectability is limited because only bivariate conditional variables are used. Notably, while the mapping is not fully position determinate, the locations of observables mapped to by specific variables cannot change unrestrictedly; hence the relocatability of building blocks is heavily constrained.

Green Ant Colony Programming (gACP) [25] is very different from bACP, despite the similarity of names. The search space is predefined by a random graph containing (multiple copies of) nodes labeled by the function and atom symbols. This graph is explored in parallel by ants which traverse the graph, cooperating to generate a tree expression—new ants being initiated whenever an encountered function has arguments that need to be instantiated. Each ant is programmed with an inherent minimum and maximum depth that it will endeavour to satisfy. Other than that, its sampling is controlled by the probability model (pheromone). Because of the use of the random graph to underly the probability model, the operation of the model is not determined by either position or context. Some relocatability of building blocks is supported, but the degree is very much determined by the structure of the specific random graph.

Grid based Ant Colony System (GACP) uses a predefined 2D grid to represent the search space [93]. Each row contains a list of all available GP tree symbols. Ants start from the first row, selecting one symbol in each row as they move downward. They stop when the sequentially selected symbols create the prefix form of a correct GP tree. The probability model (pheromone distribution) consists of a random variable for each element of every row. These independent variables learn multinomial distributions—which symbol to select for the next step. However reaching a specific variable is conditional on reaching the variable in the preceding row, so that the model representation can be viewed as a linear chain of conditional random variables. From a representational perspective, the system is fairly similar to bACP, sharing its characteristics.

Dynamic Ant Programming (DAP) resembles some of the EDA systems more than other ACO-GPs, in that it learns model structure [112]. The probability model (pheromone table) is composed of variables mapped to nonterminals as in sSG-GP. It learns from the frequencies of transitions, but its sampling step is not determined solely by the probability, because ants are prohibited from re-visiting a symbol. The probability distribution thus depends on all values in ancestor contexts, in the extreme allowing the distribution to become degenerate. To learn the structure, this system is able to insert or delete symbols in the table. It deletes symbols whose pheromone level (i.e. probability) falls below a threshold.

6.3.2 Prefix notation derivation tree genotypes

Generalised Ant Programming (GAP) adopted a CFG for model representation [42]. The representation is a tree where nodes are labeled by derived expressions. The root is labeled with the starting symbol, and each node is labeled by an expression derived by applying one production to its parent (although not explicitly stated, it appears from examples that leftmost expansions are always used). In the implemented system, probabilities are attached to full derivations rather than to derivation steps, though the possibility of attaching probabilities instead to individual derivation steps is acknowledged. Even in this amended form, the representation differs substantially, nonetheless, from the direction in most EDA systems, in which probabilities are applied to productions, rather than specific derivation steps. From another perspective, in the amended GAP, the probability of a production is determined by the full ancestral context. As a result, there is no provision for relocatability of building blocks. It can be viewed as an extreme case of PEEL, in which the learnt context of every production has been refined to the full ancestral context.

Enhanced GAP (EGAP) extended details of the learning algorithm of GAP (adapting the pheromone update strategy and path termination [97]) without changing any of the representational aspects.

Grammar Based Ant Programming (GBAP) [72] resembles DAP in assigning probabilities (pheromone) to transitions rather than paths. As in DAP, the probability table is supplemented by additional information in determining the next transition. In making this decision, GBAP also takes into account the number of possible successor states and the information gain (a quantity which is available because DAP is restricted to classification tasks, though the authors note that it could be extended to other tasks). At least at the representation level, it strongly resembles DAP, and thus sSG-GP.

6.3.3 Other linear genotypes

Ant Tree Adjoining Grammars (AntTAG) [1] represented distributions over trees using stochastic tree adjoining grammars (TAG) [38] instead of SCFG. A TAG is somewhat similar to a CFG, but its derivation mechanism is different. From a CFG, it defines elementary derivation trees, called α and β trees. In derivation, growing combinations of trees are expanded by adjunction, which inserts a β tree at a matching internal node (for a more detailed description of TAGs and their relationship with GP, please see [68]). AntTAG attempts to learn which β tree to adjoin at each potential adjunction point (in terms of what nonterminal labels that point). Unlike the previously-listed linear Chomsky grammar systems, AntTAG is linear not because it uses a linearisation (preorder) on an inherently tree structure, but because it was restricted to linear derivation trees. Whereas a restriction to linear Chomsky grammar trees severely limits the available languages (to regular languages), linear TAG derivation trees still provide reasonable representational power.

All adjunction points with the same label share the same distribution over β trees, so there is some similarity to sSG-GP; however the greater representational power of TAGs (compared to CFGs) permit it to learn slightly more complex structures. Like sSG-GP, AntTAG is position-indeterminate, but context-determinate.

Bayesian Automatic Programming (BAP) [91] uses a linear representation for CFG derivation trees; unlike the preceding systems, it does not use a prefix notation representation, but instead uses a coding derived from that of the well-known grammatical evolution [73], in which the individual is represented as a list of integers. It uses the algorithm of BOA [83] to learn dependencies between the positions in this list. However this combination leads to a complication: GE genotypes can vary greatly in length. GE implementations generally simply allow short genotypes, ignoring the remainder of the (fixed-length) string, and handle over-long individuals, that have not been completed by the time the string is exhausted, by wrapping or other strategies. Both are likely to lead to noisy learning in an EDA, so BAP uses an alternate solution, which is to discard overly short or long individuals.

BOA Programming (BOAP) [56] uses a novel linearised GP representation, the zigzag tree. Only limited information on the representation is available, however it is clear from comments in the paper that zigzag trees can be of variable length. The system learns a model using BOA [83], however we have been unable to determine how BOA is adapted to learn from variable-length strings. The system is interesting in that, having chosen a representation which is fairly position-determinate, it incorporates a low level of crossover to permit the relocation of building blocks.

N-gram GP(N-gram GP) [88] uses a linear (assignment expression) representation as its genotype.Thus it is free to take a classic N-gram approach to distribution modelling.In the description, the system used N = 3. Since an N-gram can be viewed as a depth $N - 1$ ancestral context in a linear system, N-gram GP is determinate in the $N - 1$ ancestral context. Since that context can occur anywhere in the string, it is not position-determinate. Detectability varies according to the value of N. If it is large, complex dependencies can be found; if small, many dependencies will be missed. In particular, an N-gram model may in theory have difficulty in handling long-distance dependencies; since these potentially exist in GP problems, there is some risk of limiting the problems that can be handled. On the other hand, natural language also incorporates long-distance dependencies, yet N-gram based systems such as “google translate” have nevertheless proven extraordinarily successful and robust.

Cartesian Genetic Programming with EDA (CGP-EDA) [17] is an originally-unnamed combination of EDA methods with Cartesian genetic programming, another linear representation for GP. In Cartesian GP, the genome consists of a fixed number of tuples, each tuple encoding an operation and backward references to its inputs. CGP-EDA used a simple univariate model over this genotype space, so that its primary innovation lay in extending EDA methods to CGP. In terms of the natural positional representation of CGP, CGP-EDA thus uses a fully position-determinate model.

6.4 Hybrid PMB-GP systems

The first two hybrid genetic-PMB methods described in this subsection bear somewhat the same relationship to GP and PMB-GP as does the well-known Covariance Matrix Adaptation Evolution Strategy (CMAES) [29] to evolution strategies and EDAs. That is, the outer algorithm is individual-based, but the variation operators are biased by a local probabilistic model.

Context Free Grammar Refinement (CFGR) (a name we assigned for an unnamed extension of Whigham's CFG-GP system [123]) can be viewed as a precursor for grammar-based PMB-GP. It extends a typical CFG-GP system by learning a bias toward the subregion composed of observed (i.e. selected) individuals. Among other mechanisms, it uses a method, "merit selection", which evaluates the relative importance of productions, and increases the probability of selecting productions of higher merit. In this, it pre-figured the subsequent work on EDA and ACO systems based on grammars. It also incorporates mechanisms for condensing derivation steps in individuals, thus anticipating subsequent grammar-learning systems. However it does not have a separate sampling step—information was transmitted between generations both through the populations and the grammar model—and thus cannot be seen as a true PMB-GP system.

Meta-Optimising Semantic Evolutionary Search (MOSES) [55] can be viewed as a hybrid EDA-hillclimbing search algorithm (though the unusual description somewhat obscures this). The hillclimber searches a set of demes, where a deme is a region neighbouring a promising exemplar solution. Each deme is initially sampled via a distance-dependent sampling mechanism, centred around the exemplar. Selection is then performed, and hierachical BOA (hBOA [82]) is used to learn a model, with learning repeated in an EDA-style process (differing from a classical EDA-GP in that the distribution is biased toward the exemplar). When a new solution is found that outperforms previously-found solutions, a new deme centred on it is created, possible displacing previous demes. In the inner (PMB) search, the genotype is a string of differences from the exemplar; however it differs very substantially from other string-genotype EDAs, and cannot easily be described in terms of our analysis. Because the genotype is based on the exemplar, it is difficult to extend the earlier-described conceptions of location- or context-determinacy.

Probabilistic Adaptive Mapping Developmental GP (PAM-DGP) [125] is very different from the preceding systems (and indeed, most GP systems) in being developmentally based. It uses probabilistic methods to adapt the genotype-phenotype mapping in an otherwise standard developmental GP system.

7 Issues in PMB-GP research

It will be apparent, especially from seeing the wide variety of disparate systems presented in Sect. 6, that PMB-GP systems have over the last decade seen an exuberant profusion of different ideas and implementations. This is desirable, and it is to be hoped that it will continue. However it has, to date, been supplemented by only a limited degree of consolidation and systematisation.

Most systems have been presented with only a few comparative experiments, usually on "toy" benchmark problems, and generally only in comparison with classical GP. Comparisons between PMB representations, either at genotype or model level, are rare, as are treatments of different learning algorithms. Tests on large-scale problems are also rare, leading to concerns about the scalability of PMB-GP. Despite the oft-claimed potential for theoretical analysis of PMB systems, especially the more theoretically-based EDA systems, there is little currently to point to. We discuss some directions in which PMB-GP research might go, to provide better understanding of the behaviour of PMB-GP systems.

PMB methods present new opportunities as well: in particular, the use of a statistical model inherently provides new ways to interact with knowledge, not present in genetics-based systems. These interactions can go in multiple directions. In the forward direction, the statistical model provides new opportunities for human prior knowledge to influence search. Conversely, the learning method updating the statistical model potentially provides new knowledge, not necessarily available in the population of an otherwise-equivalent genetic system. Finally, bidirectional interactions can potentially enhance the behaviour of a system. This can happen in at least two ways. As an interactive learning system, the human user may incorporate additional prior knowledge into future runs as a result of what has been learnt in earlier [109]. The simplicity and regularity of statistical models (by comparison with the populations they model) may facilitate this. In addition, there is potential for extracting generalisable knowledge from the statistical model, thus facilitating transfer learning [102], especially in the form of hyperheuristics [16]. We consider some of these issues in more detail below. Some are applicable not only to PMB-GP, but more widely to PMB systems in general.

7.1 PMB-GP systems and GP analysis

7.1.1 Analysing the behaviour of conventional evolutionary algorithms

One important justification given for the study of EDAs in particular is that their potentially more tractable behaviour may cast light on the effectiveness of evolutionary algorithms. This justification applies equally to GP (or perhaps, in light of GP's very limited theoretical analysis, with even more force). Unfortunately this promise has not, to date, been fulfilled.

In this respect, the recent study of Hemberg et al. [35] is highly encouraging: a study of the evolution of statistical relationships in GP (itself informative for the understanding of GP behaviour) led to a new statistical model for EDA-GP, and a new system (OFGP). Unfortunately OFGP fails to perform competitively with GP (itself informative, in that it tells us that the OFGP N-gram model did not fully capture the important behavioural information). The authors suggest that this is because their model did not capture sibling relationships (which GP operators also preserve). In addition, however, the analysis could only capture fixed-length positional relationships (for example, the commonality in the relationships between '+' and 'X' in '+ X' and in '+ + 0 X' could not be recognised). We note that this could explain the relatively poorer performance of OFGP (i.e. in that it failed to capture important relationships), the tendency to lose good behaviours once evolved (in that the model might not capture the important components of those behaviours) and the tendency to favour small solutions (in that the system would be more likely to capture short relationships than long, because the short relationships would be less likely to be disrupted by introns).

7.1.2 Analysing the effects of different genotype representations

To date, there has been only limited study of the effects of genotype representations. One exception is our own work, which recently investigated the effects of sampling bias in PPT genotypes [43, 44], demonstrating that sampling bias is a serious issue for at least some genotype representations, and proposing a solution. More such studies are clearly needed, for both the sampling and learning phases over different genotype representations.

More generally, comprehensive comparative studies are also needed, experimentally testing the performance of different genotype structures (and if possible, the underlying reasons for any differences).

7.1.3 Analysing the effects of different probability models

It is almost a truism in PMB-GP research to date that streams of research have begun with simple univariate models and moved on to fixed multivariate models. In many cases, there has been a further extension, to learning the model structure. In some cases, this has been based on rigorous comparisons, but in others more on opinion. More study of these issues is needed, as is a clearer framework (to which we hope this study makes a first contribution) for understanding these issues across a range of genotype representations..

7.1.4 Analysing and comparing different algorithm components

This study has largely focused on representation issues—especially, genotype representation and probability model structure. However a PMB-GP system is not just its representations. There is a need for more detailed study of the effects of at least:

1. Different sampling strategies—most systems use some form of probabilistic logic sampling; a more thorough study and comparison with alternative sampling strategies is needed.
2. Different probability update strategies—one of the major areas of distinction between EDA and ACO systems. Most current EDA systems use maximum likelihood update, while ACO systems use less formalised strategies. Both are generally adapted, in somewhat ad-hoc ways, to increase diversity in recognition of the risk of loss of diversity in iterated sampling/learning cycles. Better theory (or perhaps closer attention to the already-developed theory in the more general EDA and ACO fields) is needed.
3. Different structure-learning strategies—a number of systems attempt to learn not merely the probabilities within a fixed probability model, but also the probability model itself. There are a wide range of issues here.
   One key issue is how to trade off model complexity against model accuracy. This is often reduced to maximising the posterior probability of sampling fit individuals in the next round—a classic machine learning question. But the learning round is repeated in an iterative algorithm. Repeated maximisation is

likely to amplify sampling and other biases, leading to sub-optimal outcomes. Somewhat greater generality than would be predicted by machine learning heuristics is desirable—but how much more general? There does not seem to be any current theory for this.

Another is the form of learning. Both bottom-up (generalising from the selected sample upward) and top-down (specialising from the previous generation's model toward the current selected population) have been proposed. Bottom-up learning increases the risk of overfitting the current population, while top-down algorithms often lead to greater algorithmic complexity. More complex, multi-directional algorithms are also possible.

7.2 Knowledge and PMB-GP systems

7.2.1 Incorporating user knowledge of the form of solutions

As in genetics-based systems, some representations (notably grammars) provide the opportunity to incorporate user knowledge in the search process, by excluding potential solutions that do not meet those constraints. However PMB-GP systems provide additional opportunities, in particular by permitting uncertain knowledge to be incorporated through the probability model. This can be achieved either through the standard statistical approach of specifying a statistical prior, or through structural control to alter the available solution space [120, 127]. It may seem that this differs little from the classical GP approach, of biasing the initial population. Indeed this is correct in the case of PMB-GP systems that learn the probability model anew at each generation. However most systems incorporate some form of incremental learning, thus allowing PMB-GP systems to use lower overall levels of bias, and protecting them from the worst consequences of incorrect biases. However the effects of user-imposed biases in PMB-GP systems have not been carefully studied; such study is now overdue.

7.2.2 Incorporating user knowledge about properties of the search space

Grammars were originally introduced into GP to allow the user to incorporate knowledge about the search space [127, 120]. Grammar-based PMB-GPs inherit this ability.

The explicit probability model provides other opportunities to incorporate knowledge. For example, GP search spaces typically involve numerous symmetries [101]: in boolean and arithmetic search spaces, many logical and arithmetic identities generate symmetries. In GP, these symmetries are more a hindrance than a help—in fact, we rely on spontaneous symmetry breaking to permit GP systems to converge to a few neighbouring solutions, among the myriad actual solutions that most problems have. There are alternatives—many other search methods rely on canonicalisation to factor out the symmetries, reducing the search space, often to an extent exponential in the problem size. This generally works poorly in GP, because canonicalisation reduces diversity and thus limits productive exploration. PMB-GPs do not face the same problem, because canonicalisation can be imposed on the selected population just prior to the learning phase (thus giving the search-space-reduction advantages of canonicalisation), but then removed through random choice of symmetries during sampling, prior to selection, giving the best of both worlds. Investigation of the role of symmetries in EDA-GP thus seems well worth the effort.

7.2.3 Extracting knowledge from the final model

The final model incorporates information from the final population, but it also encapsulates information from previous stages of the evolution. In addition, in structure-learning systems, it incorporates knowledge about the important components, and the relative importance of those components, that may not be available in a genetically-evolved population. If the aim of our PMB-GP application is simply to find an optimal solution, this information may not be useful. However in many applications in scientific study and design, generating new knowledge about the domain may be much more important. In this respect, it will be important to determine both the extent to which useful knowledge can be extracted from such models, and the value of so doing. In such studies, it will be important—even more so than in optimisation—to preserve diversity, so that there can be reasonable confidence that the model covers most or all good solutions; mechanisms for such enhanced diversity preservation need exploration. In particular, the role of symmetries and symmetry-breaking (or its avoidance) need investigation.

7.2.4 Knowledge re-use

There are two important roles for knowledge re-use in PMB-GP: in iterative learning and in transfer learning. In iterative learning, the user may examine the information generated in one round of application of PMB-GP (by examining the resulting model). This information may well reveal the user's further unconscious knowledge that may be incorporated into future runs of the system. This strategy has been previously used with standard GP [121, 109], but the probability model of a PMB system provides a more sophisticated means.

Transfer learning is the generalisation of knowledge gained in one problem to other related problems, in order to speed up learning [19, 27, 51, 78, 117]. Hyperheuristics [16] are the main mechanism used to date in evolutionary algorithms. PMB systems, especially those which learn model structure, can potentially provide a more extensive knowledge transfer than is available in classical hyperheuristic systems, so long as the knowledge embedded in the probability model can be extracted and generalised to new problems. Research in this area has been the subject of some recent research in EDAs [54, 113, 86, 102], but has a very high potential pay-off for PMB-GPs as well.

7.3 Theory-based extensions

7.3.1 Using more information from evaluation

In most PMB systems (for optimisation as well as for GP problems) truncation selection is used. As a result, each individual provides a maximum of one bit of information in return for the cost of evaluation. Truncation selection treats an individual which barely passes the truncation cutoff the same as an elite individual, and one which just fails the same as the worst individual ever evaluated. Recognition of this issue in PMB optimisation, has led to algorithms using the Boltzmann distribution, or otherwise taking the fitness value into account [99] . Despite the typically much higher evaluation cost in PMB-GP (and hence the greater importance of using all the information resulting from evaluation), we are not aware of any extensions of such methods to GP problems. It would be highly desirable to evaluate their potential.

7.3.2 Making use of negative instances

A restricted form of the above would at least make use of the negative instances for learning—the knowledge that a particular area is unlikely to contain a solution may be of equal importance to positive knowledge. Negative information has rarely been used in PMB systems—to the best of our knowledge, only in the Boltzmann-style models just described, and in Michalski's somewhat EDA-like system LEM [60, 61]. However it is worth noting the importance of negative instances even in learning from crisp instances. Numerous problems that are not learnable from positive instances alone, are learnable if negative instances are provided [118].

7.3.3 Learning, parsimony and noise

The close relationship between generality, parsimony and noise have been heavily studied in machine learning, and by now are relatively well understood [9, 92, 119]. As we already noted, this understanding breaks down when applied to PMB systems, because the level of generalisation that is appropriate for a single round of learning may not be appropriate (in general, is likely to be overfitted to chance fluctuations) when embedded in an iterated sampling/learning cycle.

However with PMB-GP there is a further complication. Most real-world GP problems are themselves learning problems, in that we have a set of samples of data, to which we wish to fit a GP solution. The samples of data are a somewhat randomly selected subset of all possible samples, and we hope to generate a model which will fit future samples, which we hope will be generated from the same distribution. Thus in effect, we have two learning systems embedded one within the other (an outer layer trying to learn a GP solution, which we hope will generalise to new instances, and an inner layer trying to learn a good probability model from candidate GP solutions). These two systems interact with each other in ways that immensely complicate the underlying theory. What is more, the outer layer, learning from instances in a complex fitness landscape, faces a further complication. The "parsimony" landscape is smooth and easy to learn, while the fitness landscape is not. Hence simple approaches to trading these off inside an evolutionary algorithm using theoretically-derived trade-off parameters often fail. They converge to an over-parsimonious local optimum before the fitness landscape search has a chance to get under way.

In PMB-GP, there is a further complication: there is a built-in stochastic bias toward smaller individuals. It results from both sampling bias and selection bias. In sampling, the more complex an individual is, the more difficult it is to generate in sampling process. In selection, the more complex an individual is, the more likely a particular (good or bad) component is to be evaluated in a deceptive context that misleads the PMB algorithm about its quality [43, 44] (this analysis is focused on PPT models, but the underlying arguments extend generally to PMB-GP models). In contrast to the bloat in genetically-based GP systems, this parsimony pressure may seem desirable (and has been repeatedly argued as such in PMB-GP papers). However the extent of the bias is not tuned to the particular problem (for example, a properly tuned parsimony pressure should depend on the level of noise in fitness evaluation, while the PMB-GP biases are independent of this).

Overall, there is much that we do not understand about PMB-GP systems as learning systems; a better linkage with machine learning theory is desirable, however difficult it may be to accomplish in fact.

7.4 Practical use of PMB-GPs

7.4.1 Applications to real-world problems

Application of the PMB-GP research to real world problems is crucial to win wider acceptance. So far, most research has focused on developing theoretically well-founded models rather than practical applications. Testing has generally used toy benchmark problems such as royal tree, MAX, DMAX, symbolic regression [33, 34, 88, 106] and so on. The results show significant performance improvements relative to conventional algorithms, implying good potential for real world problems. There have been few practical experiments. CSGT was proposed to evolve Snake bot's locomotion in various environments [114], and GNP-EDA was applied to rule mining from traffic data [52] and Khephera robot control [53]. Further such applications are crucial to the future development of the field.

7.4.2 Preferred problems

It is likely that PMB methods are more suited to some GP problems than others; it is highly unlikely that PMB methods can out-perform genetic methods on all GP problems. Hence characterising the relative performance of PMB and genetic methods on different domains GP would be highly desirable. However it is unlikely to be easy—it is very limited even for EDA domains, where the comparison should be much easier. Similar considerations apply within PMB-GP: which systems are best suited to which problems? This issue has, to date, barely been touched on.

At present, the basis for choosing PMB-GP over classical GP is more about other aspects of the problem than the domain. If the user has probabilistic knowledge to inform the search, PMB methods are more likely to be able to incorporate it. If the user desires explanatory models, the final probability model of the PMB-GP system may give better guidance than is available simply from the best solution.

7.4.3 Open-source systems

There are a number of widely available EDA systems, both general libraries such as MATEDA [100] and implementations of specific algorithms [79–81, 103]. To the best of our knowledge, there is currently no equivalent freely-available, industrial-strength implementation of a PMB-GP system. This is primarily because current PMB-GP systems are still research software, subject to repeated tweaking. There is a clear need for a flexible, modular PMB-GP system, freely available and built to software industry quality standards. Such a system would doubtless provoke a blossoming of research and application of EDA-GP, and perhaps lead to clarification of many of the issues raised here.

7.5 Disclosure

Throughout this paper, we have endeavoured to present a neutral perspective on a wide range of matters. It is unlikely we succeeded—it is almost inevitable that our views have influenced the way we present issues, and even the selection of issues (because our views are reflected in the importance we attach to specific issues). Rather than pretend that our discussion could be completely objective, we here present our own views on representation issues in EDA-GP, together with the arguments for them, so that the extent to which they have influenced the rest of the presentation can be independently assessed.

7.5.1 Tree versus other representations

All our work in GP and EDA-GP has used tree-based representations. The arguments for and against tree-based representations have been rehearsed in many places [6, 18, 26, 39, 47, 69, 116, 124]. We don't have strong views on this; our neglect of linear and graph representations reflects pragmatic considerations (once you have built software for tree representations, it is easier to continue with it). Nevertheless, our experience with tree representations may have subconsciously affected our presentation.

7.5.2 Linearised tree versus direct tree representations

On this issue, we do have strong views, especially as regards PMB systems. Whatever may be the case with GP, and the ability of crossover to handle complex search spaces, we believe that there is a clear argument with respect to PMB systems. Dependences within trees are highly likely to reflect the tree structure (after all, the belief that the mapping between tree space and fitness is likely to be continuous is exactly what justifies a preference for heuristic rather than pure random search). If dependences reflect the tree structure, then linearisation simply complicates the learning phase of a PMB-GP (it needs to rediscover the tree relationships for itself, as well as discovering the inherent relationships), without providing any corresponding benefit. We note that similar issues arise for tree-based

GP representations that do not use the tree structure during learning, even if the tree structure is explicitly represented.

7.5.3 Relocatability of building blocks

We are convinced that relocatablity is crucial in GP. This reflects Koza's argument for its importance [47], that human programmers deal with blocks of code, and expect them to be relocatable. It also reflects the subsequent repeated verification, over many years of experimentation, of the importance of subtree crossover and its derivatives as search mechanisms in GP [70, 71, 115]. Finally, it also reflects the experience, over many years, of the importance of introns and near-introns in GP, since in the presence of introns, relocatability of the meaningful components of GP is essential to their operation.

7.5.4 Importance of hidden relationships

We are of the view that important probabilistic relationships in specific problems may rely on components that are not explicit in the genotype (otherwise, we are expecting of the user who defines the genotype an almost god-like foreknowledge). Thus good representations for probabilistic models almost inevitably require some mechanism to discover and represent such components. We expect models supporting, and able to discover, some such hidden variables to be more successful. We note that grammar-based systems able to discover new nonterminals or to learn new annotations provide such mechanisms, as do dependency models in systems such as ECGP.

7.5.5 Importance of user-imposed biases

GP search spaces, especially for real-world-scale problems, are already extremely difficult . Thus we need to take advantage of any edge that we can find. In most domains, users already have substantial prior knowledge about the search space—about the likely form of solutions, about the relationships between good solutions, and about the acceptable forms of solutions (in many real-world domains, solutions of high fitness may nevertheless not be acceptable; examples include solutions whose value cannot be computed when applied to real-world examples, and solutions lacking specific formal—algebraic or analytic—properties).

8 Conclusion

We have presented what we hope is a critical and comprehensive survey of the application of probabilistic model building systems to GP problems. The survey brings together two main strands of research effort (EDA-GP and ACO-GP), showing their commonalities and differences. It also explores contributions from classical EDA and ACO research and that from grammar induction. In analysing systems, we have emphasised a number of factors that we view as critical: modularity, positional and context determinacy/dependence, and the related issue of structural learning. We used these factors as guides in analysing different PMB-GP systems.

In the past decade, PMB-GP work has proliferated in a number of different research streams, with interaction between them ranging from almost zero to substantial. As a result of the implicit complexity of GP search spaces, this research is still in its infancy in comparison with that into EDA and ACO. The underlying reason for this complexity is the requirement to learn relational information from GP trees. While EDA-GP was inspired by EDA, and similarly for ACO-GP, both are underlain, implicitly or explicitly, by probabilistic graphical models, which are well developed for propositional search spaces. However to be successful, PMB-GP systems very probably require relational learning, leaving them in the domain of the much more complex—and much less explored—field of stochastic relational learning (SRL) [22].

Probabilistic model building has major potential in genetic programming. The exact form that eventually succeeds is difficult to predict, but we hope that the systematisation we have attempted here, and the open questions we have raised, can help to stimulate the field and lead to more focused research. We also hope that, by surveying across the range of EDA- and ACO-based systems in a common framework, we have contributed to bringing these two fields together, and that greater cross-fertilisation between them may result. More generally, we hope that drawing attention to the potential of the field, and to the sheer number of open questions, can stimulate greater interest in and a more rigorous approach to the underlying issues.

Acknowledgments This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Project No. 2012-004841). Xuan Hoai Nguyen was partly funded by The Vietnam National Foundation for Science and Technology Development (NAFOSTED), under Grant Number 102.01–2011.08, for doing this work. The ICT at Seoul National University provided research facilities for this study.

Index

References

1. H.A. Abbass, X. Hoai, R.I. McKay. AntTAG: A new method to compose computer programs using colonies of ants. In Proceedings of the 2002 IEEE Congress on Evolutionary Computation, vol. 2, Honolulu, HI, USA. (IEEE Press, New York, 2002), p. 1654–1659
2. P.J. Angeline. Subtree crossover: Building block engine or macromutation? In Genetic Programming 1997: Proceedings of the Second Annual Conference, pages 9–17, Stanford University, CA, USA, 13–16 July 1997. Morgan Kaufmann
3. P.J. Angeline, J.B. Pollack. Coevolving high-level representations. In Artificial Life III, vol. XVII of Santa Fe Institute Series, pages 55–71, Santa Fe, New Mexico, USA, 15–19 June 1994. Addison-Wesley, USA
4. S. Baluja. Population-based incremental learning: a method for integrating genetic searching based function optimization. Technical Report CMU-CS-94-163, Computer Science Dept, Carnegie Mellon University, Pittsburgh, PA, USA, 1994
5. W. Banzhaf. Genetic programming for pedestrians. In Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93, p. 628, Urbana-Champaign, IL, USA, 17–21 July 1993. Morgan Kaufmann
6. W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francone. Genetic Programming: An Introduction; On the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann, San Francisco, 1998
7. H.G. Beyer, An alternative explanation for the manner in which genetic algorithms operate. BioSystems **41**, 1–15 (1997)
8. M. Birattari, G. Di Caro, and M. Dorigo. Toward the formal foundation of ant programming. In Proceedings of the Third International Workshop on Ant Algorithms, volume 2463 of Lecture Notes in Computer Science, pages 188–201, Brussels, Belgium September 2002. (Springer, Berlin), p. 12–14
9. C.M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics). (Springer, New York, Inc., Secaucus, 2006)
10. M. Boryczka, Eliminating introns in ant colony programming. Fundamenta Informaticae **68**(1-2), 1–19 (2005)
11. M. Boryczka. Ant colony programming with the candidate list. In Proceedings of the Agent and Multi-Agent Systems: Technologies and Applications, Second KES International Symposium, KES-AMSTA 2008, volume 4953 of Lecture Notes in Computer Science, pp. 302–311, Incheon, Korea, 26–28 March 2008. (Springer, Berlin)
12. M. Boryczka, Z.J. Czech. Solving approximation problems by ant colony programming. In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002. (Morgan Kaufmann Publishers), p. 133
13. P.A.N. Bosman, E.D. de Jong. Grammar transformations in an EDA for genetic programming. In Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at

the Genetic and Evolutionary Computation Conference—GECCO-2004, Seattle, Washington, USA, June 2004. (Springer, Berlin)
14. P.A.N. Bosman, E.D. de Jong. Learning probabilistic tree grammars for genetic programming. In Parallel Problem Solving from Nature - PPSN VIII, volume 3242 of Lecture Notes in Computer Science Birmingham, UK, Sep 2004. (Springer, Berlin), p. 192–201
15. P.A.N. Bosman, D. Thierens. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference (GECCO-2000), Las Vegas, Nevada, USA, 8–12 July 2000. (Morgan Kaufmann, Burlington), p. 197–200
16. E.K. Burke, M.R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J.R. Woodward. Exploring hyper-heuristic methodologies with genetic programming, volume 1 of Intelligent systems Reference Library, chapter 6 Springer, 2009, pp 177–201
17. J. Clegg. Combining cartesian genetic programming with an estimation of distribution algorithm. In GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA, 12-16 July 2008. ACM, p. 1333–1334
18. 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, Carnegie Mellon University, Pittsburgh, PA, USA, 24-26 July 1985, p. 183–187
19. A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, Symbolic knowledge extraction from trained neural networks: a sound approach. Artif. Intell. **125**(1-2), 155–207 (2001)
20. A.P Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Statis. Soc. Series B (Methodological) **39**(1), 1–38 (1977)
21. M. Dorigo, T. Stützle, Ant Colony Optimization. (MIT Press, Cambridge, MA, 2004)
22. L. Getoor, B. Taskar, Introduction to Statistical Relational Learning. (The MIT Press, Cambridge, 2007)
23. A. Geyer-Schulz, Fuzzy Rule-Based Expert Systems and Genetic Machine Learning, volume 3 of Studies in Fuzziness. (Physica-Verlag, Heidelberg, 1995)
24. D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. (Addison-Wesley, USA, 1989)
25. J. Green, J.L. Whalley, C.G. Johnson. Automatic programming with ant colony optimization. In Proceedings of the 2004 UK Workshop on Computational Intelligence, UK, 6–8 September 2004. (Loughborough University, Loughborough), pp. 70–77
26. F. Gruau. Genetic synthesis of modular neural networks. In Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93, Urbana-Champaign, IL, USA, 17-21 July 1993. (Morgan Kaufmann, Burlington), pp. 318–325
27. P. Haddawy. Generating bayesian networks from probability logic knowledge bases. In Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, Seatle, WA, USA, 29-31 July 1994. (Morgan Kaufmann Publishers Inc, Burlington) pp. 262–269
28. S. Handley. On the use of a directed acyclic graph to represent a population of computer programs. In Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, vol. 1, pp. 154–159, 1994
29. N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In Evolutionary Computation, 1996., Proceedings of IEEE International Conference on. IEEE, 1996, pp. 312–317
30. G. Harik, F. Lobo, and K. Sastry. Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA), volume 33 of Studies in Computational Intelligence. (Springer, Berlin, 2006), pp. 39–61
31. Y. Hasegawa and H. Iba. Estimation of Bayesian network for program generation. In Proceedings of the Third Asian-Pacific workshop on Genetic Programming, pp. 35–46, Military Technical Academy, Hanoi, VietNam, 2006
32. Y. Hasegawa and H. Iba. Estimation of distribution algorithm based on probabilistic grammar with latent annotations. In Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007. IEEE Computational Intelligence Society, IEEE Press, pp. 1043–1050
33. Y. Hasegawa, H. Iba, A Bayesian network approach to program generation. IEEE Trans. Evol. Comput. **12**(6), 750–764 (2008)
34. Y. Hasegawa, H. Iba, Latent variable model for estimation of distribution algorithm based on a probabilistic context-free grammar. IEEE Trans. Evol. Comput. **13**(4), 858–878 (2009)

35. E. Hemberg, K. Veeramachaneni, J. McDermott, C. Berzan, and U.M. O'Reilly. An investigation of local patterns for estimation of distribution genetic programming. In GECCO '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference. ACM, 7-11July 2012, pages 767–774
36. H. Iba, Y. Hasegawa, T.K. Paul, Genetic Programming and Machine Learning. (CRC Complex and Enterprise Systems Engineering. CRC Press, Boca Raton, 2009)
37. C.Z. Janikow. Adapting representation in genetic programming. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), volume 3103 of Lecture Notes in Computer Science. (Springer, Berlin , 2004), pp. 507–518
38. A.K. Joshi, L.S. Levy, M. Takahashi, Tree adjunct grammars. J. Comput. Syst. Sci. **10**(1), 136–163 (1975)
39. W. Kantschik and W. Banzhaf. Linear-tree GP and its comparison with other GP structures. In Genetic Programming, Proceedings of EuroGP' 2001, volume 2038 of Lecture Notes in Computer Science, Lake Como, Italy, 18–20 April 2001. Springer, Berlin, pp. 302–312
40. H. Katagiri, K. Hirasama, and J. Hu. Genetic network programming-application to intelligent agents. In 2000 IEEE International Conference on Systems, Man, and Cybernetics, vol 5. IEEE, 8-11 October 2000, pp. 3829–3834
41. H. Katagiri, K. Hirasawa, J. Hu, and J. Murata. Network structure oriented evolutionary model—genetic network programming–and its comparison with. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), San Francisco, California, USA, 7–11 July 2001. (Morgan Kaufmann Publishers Inc., Burlington) p. 179
42. C. Keber, M. G. Schuster. Option valuation with generalized ant programming. In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Newyork, USA, 9–13 July 2002. (Morgan Kaufmann Publishers, Burlington), p. 74–81
43. K. Kim, B.(R.I.) McKay, D. Punithan. Sampling bias in estimation of distribution algorithms for genetic programming using prototype trees. In PRICAI 2010: 11th Pacific Rim International Conference on AI, volume 6230 of Lecture Notes in Artificial Intelligence. (Springer, Berlin 2010), pp. 100–111
44. K. Kim, R.I.(Bob) McKay, Stochastic diversity loss and scalability in estimation of distribution genetic programming. IEEE Trans. Evol. Comput. **17**(3), 301–320 (2013)
45. D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques. (The MIT Press, Cambridge, 2009)
46. J.R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Dept. of Computer Science, Stanford University, June 1990
47. J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. (MIT press, Cambridge, 1992)
48. W.B. Langdon, R. Poli, Foundations of Genetic Programming. (Springer, Berlin, 2002)
49. P. Larranaga, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation (Genetic Algorithms and Evolutionary Computation). (Springer, Berlin, 2001)
50. P. Larranaga, H. Karshenas, C. Bielza, R. Santana, A review on probabilistic graphical models in evolutionary computation. J. Heuristics **18**, 795–819 (2012)
51. P.P. Le, A. Bah, L. H. Ungar. Using prior knowledge to improve genetic network reconstruction from microarray data. In Silico Biology, 4(27), 2004
52. X. Li, S. Mabu, H. Zhou, K. Shimada, and K. Hirasawa. Genetic network programming with estimation of distribution algorithms and its application to association rule mining for traffic prediction. In Proceedings of ICROS-SICE International Conference, 2009, Fukuoka, Japan, 18–21August 2009. IEEE, pages 3457–3462
53. X. Li, S. Mabu, H. Zhou, K. Shimada, and K. Hirasawa. Genetic network programming with estimation of distribution algorithms for class association rule mining in traffic prediction. In Proceedings of 2010 IEEE Congress on Evolutionary Computation (CEC 2010), Barcelona, Spain, 18–23 July 2010. (IEEE Press, New York) pp. 37–44
54. X. Liu, Y. Wu, and J. Ye. An improved estimation of distribution algorithm in dynamic environments. In 2008 Fourth International Conference on Natural Computation(ICNC'08), volume 6, p. 269–272, 2008
55. M. Looks. Scalable estimation-of-distribution program evolution. In Proceedings of the 9th annual conference on Genetic and evolutionary computation, volume 1 of GECCO '07, London, UK, 7–11 July 2007. (ACM Press, New York) pp. 539–546

56. M. Looks, B. Goertzel, and C. Pennachin. Learning computer programs with the Bayesian optimization algorithm. In Proceedings of the 2005 conference on Genetic and evolutionary computation, volume 1 of GECCO '05, pp. 747–748, Washington DC, USA, 25–29 June 2005. ACM Press, New York
57. C.D. Manning, H. Schutze, Foundations of Statistical Natural Language Processing. (MIT Press, Cambridge, 1999)
58. T. Matsuzaki, Y. Miyao, and J. Tsujii. Probabilistic CFG with latent annotations. In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), Ann Arbor, Michigan, USA, 25-30 June 2005. Association for Computational Linguistics., pp 75–82
59. N.F. McPhee, B. Ohs, T. Hutchison. Semantic building blocks in genetic programming. In Proceedings of the 11th European conference on Genetic programming, EuroGP 2008, volume 4971 of Lecture Notes in Computer Science, Naples, Italy, 26-28 March 2008. Springer, Berlin p. 134–145
60. R.S. Michalski, Learnable evolution model: Evolutionary processes guided by machine learning. Mach Learn **38**, 9–40 (2000)
61. R.S. Michalski and J. Wojtusiak. The distribution approximation approach to learning from aggregated data. Technical Report MLI 08-2, Reports of the Machine Learning and Inference Laboratory, George Mason University, 2008
62. J.F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Proceedings of the Genetic and Evolutionary Computation Conference, volume 2, Orlando, Florida, USA, 13–17 July 1999. (Morgan Kaufmann Publishers Inc., San Francisco) pp. 1135–1142
63. J.F. Miller and P. Thomson. Cartesian genetic programming. In Genetic Programming, Proceedings of EuroGP'2000, volume 1802 of Lecture Notes in Computer Science, Edinburgh, UK, 15-16 April 2000. Springer, Berlin, pp121–132
64. D.J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 7 May 1993
65. D.J. Montana, Strongly typed genetic programming. Evol. Comput. **3**(2), 199–230 (1995)
66. H. Mühlenbein, J. Bendisch, and H.-M. Voigt. From recombination of genes to the estimation of distributions: II. continuous parameters. In Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, volume 1141 of Lecture Notes in Computer Science, Berlin, Germany, 22-26 September 1996. (Springer, Berlin) pp. 188–197
67. H. Mühlenbein and G. Paass. From recombination of genes to the estimation of distributions I. binary parameters. In Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, volume 1141 of Lecture Notes in Computer Science, Berlin, Germany, 22-26 September 1996. (Springer, Berlin) pp. 178–187
68. X.H. Nguyen, R.I. McKay, D. Essam, Representation and structural difficulty in genetic programming. IEEE Trans. Evol. Comput. **10**(2), 157–166 (2006)
69. P. Nordin. A compiling genetic programming system that directly manipulates the machine code. In Advances in genetic programming, chapter 14, pp. 311–331. (MIT Press, Cambridge, 1994)
70. P. Nordin and W. Banzhaf. Complexity compression and evolution. In Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95), Pittsburgh, PA, USA, 15-19July 1995. (Morgan Kaufmann Publisher Inc., San Francisco), pp. 310–317
71. P. Nordin, F. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, Advances in Genetic Programming 2, chapter 6. (MIT Press, Cambridge, 1996), pp. 111–134
72. J.L. Olmo, J. R. Romero, and S. Ventura. A grammar based ant programming algorithm for mining classification rules. In Proceedings of 2010 IEEE Congress on Evolutionary Computation (CEC 2010), Barcelona, Spain, 18–23 July 2010. IEEE Press, pages 225–232
73. O'Neill M., Ryan C. (2001) Grammatical evolution. IEEE Trans. Evol. Comput. 5(4):349–358
74. U.M. O'Reilly. An Analysis of Genetic Programming. PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, 22 September 1995
75. U.M. O'Reilly. Investigating the generality of automatically defined functions. In Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, 28–31 July 1996. MIT Press, pages 351–356
76. U.M. O'Reilly and F. Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In Foundations of Genetic Algorithms 3, Estes Park, Colorado, USA, 31 July–2 August 1994. Morgan Kaufmann Publishers Inc. Published 1995, pages 73–88

77. A. Ortega, de la M. Cruz, M. Alfonseca, Christiansen grammar evolution: Grammatical evolution with semantics. IEEE Trans. Evol. Comput. **11**(1), 77–90 (2007)
78. S.J. Pan, Q. Yang, A survey on transfer learning. IEEE Trans. Know. Data Eng. **22**(10), 1345–1359 (2010)
79. M. Pelikan. A simple implementation of bayesian optimization algorithm (boa) in C++ (version 1.0). Technical Report 99011, IlliGAL, University of Illinois at Urbana–Champaign, 1999
80. M. Pelikan. A C++ implementation of bayesian optimization algorithm with decision graphs. Technical Report 2000025, IlliGAL, University of Illinois at Urbana–Champaign, 2000
81. M. Pelikan. Implementation of the dependency-tree estimation of distribution algorithm in C++. Technical Report 2006010, IlliGAL, University of Illinois at Urbana–Champaign, 2006
82. M. Pelikan, D.E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001), San Francisco, California, July 2001. Morgan Kaufmann Publishers Inc
83. M. Pelikan, D. E. Goldberg, and E. Cantu-Paz. BOA: The Bayesian optimization algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, volume 1, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufamann, pp. 525–532
84. M. Pelikan, D. E. Goldberg, and F. G. Lobo. A survey of optimization by building and using probabilistic models. In Proceedings of the 2000 American Control Conference, volume 5, Baltimore, MD, USA, 28–30 June 2000, pages 3289–3293
85. M. Pelikan, D.E. Goldberg, F.G. Lobo, A survey of optimization by building and using probabilistic models. Comput. Optim. Appl. **21**(1), 5–20 (2002)
86. M. Pelikan, M. Hauschild, and P. Lanzi. Transfer learning, soft distance-based bias, and the hierarchical boa. In Parallel Problem Solving from Nature - PPSN XII, volume 7491 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pages 173–183
87. Poli R., Langdon W.B. (1998) Schema theory for genetic programming with one-point crossover and point mutation. Evol. Comput. 6(3):231–252
88. R. Poli and N. F. McPhee. A linear estimation-of-distribution GP system. In Proceedings of the 11th European conference on Genetic Programming, EuroGP 2008, volume 4971 of Lecture Notes in Computer Science, Naples, Italy, 26-28 March 2008. (Springer, Berlin), p. 206–217
89. A. Ratle and M. Sebag. Avoiding the bloat with probabilistic grammar-guided genetic programming. In Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001, volume 2310 of Lecture Notes in Computer Science, Creusot, France, 29-31 October 2001. (Springer, Berlin), p. 255–266
90. C.R. Reeves, J.E. Rowe, Genetic algorithms : principles and perspectives ; a guide to GA theory. (Kluwer, Netherland, 2004)
91. E.N. Regolin, A.T.R. Pozo. Bayesian automatic programming. In Proceedings of the 8th European Conference on Genetic Programming, EuroGP 2005, volume 3447 of Lecture Notes in Computer Science, Lausane, Switzerland, 30 Mar–1 Apr 2005. (Springer, Berlin, p. 38–49)
92. J. Rissanen, Modeling by shortest data description. Automatica **14**(5), 465–471 (1978)
93. S.A. Rojas, P.J. Bentley. A grid-based ant colony system for automatic program synthesis. In Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference, Seattle, Washington, USA, 26 July 2004
94. J.P. Rosca. Analysis of complexity drift in genetic programming. In Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann Publishers Inc., pp. 286–294
95. J.P. Rosca, D.H. Ballard. Genetic programming with adaptive representations. Technical Report TR 489, University of Rochester, Computer Science Department, Rochester, NY, USA, February 1994
96. O. Roux, C. Fonlupt. Ant programming: or how to use ants for automatic programming. In Proceedings of the Second International Conference on Ant Algorithms (ANTS2000), Brussels, Belgium, September 2000, p. 121–129
97. A. Salehi-Abari and T. White. Enhanced generalized ant programming (EGAP). In Proceedings of the 10th annual conference on Genetic and evolutionary computation(GECCO '10), Portland, Oregon, USA, 7-11 July 2008. ACM New York, pages 111–118
98. R.P. Salustowicz, J. Schmidhüber, Probabilistic incremental program evolution. Evol. Comput. **5**(2), 123–141 (1997)
99. R. Santana, Estimation of distribution algorithms with Kikuchi approximations. Evol. Comput. **13**(1), 67–97 (2005)

100. R. Santana, C. Echegoyen, A. Mendiburu, C. Bielza, J. A. Lozano, P. Larraaga, R. Armaanzas, and S. Shakya. Mateda: A suite of EDA programs in Matlab. Technical Report EHU-KZAA-IK-2/09, University of the Basque Country, Feb 2009
101. R. Santana, R.I. McKay, and J.A. Lozano. Symmetry in evolutionary and estimation of distribution algorithms. In IEEE Congress on Evolutionary Computation, page To Appear. IEEE Computational Intelligence Society, (IEEE Press, New York 2013)
102. R. Santana, A. Mendiburu, and J.A. Lozano. Structural transfer using EDAs: An application to multi-marker tagging SNP selection. In 2012 IEEE Congress on Evolutionary Computation (CEC), p. 1–8, 2012
103. K. Sastry, L. de la Ossa, F.G. Lobo. χ–ary extended compact genetic algorithm for matlab in C++. Technical Report 2006013, IlliGAL, University of Illinois at Urbana–Champaign, 2006
104. K. Sastry, D.E. Goldberg. Probabilistic model building and competent genetic programming. Genetic Programming Theory and Practise, p. 205–220, 2003
105. 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 IEEE Congress on Evolutionary Computation, CEC2003, pages 1639–1646, Canberra, Australia, 8-12 December 2003. University of New South Wales, IEEE Press, New York
106. Y. Shan, R.I. McKay, R. Baxter, H. Abbass, D. Essam, H.X. Nguyen. Grammar model-based program evolution. In Proceedings of the 2004 IEEE Congress on Evolutionary Computation, Portland, Oregon, USA, 20-23 June 2004. (IEEE Press, New York), p. 478–485
107. Y. Shan, R. I. McKay, D. Essam, and H. A. Abbass. A survey of probabilistic model building genetic programming. In Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, volume 33 of Studies in Computational Intelligence, chapter 6. (Springer, Berlin, 2006), p. 121–160
108. Y. Shan, R. I. McKay, D. Essam, and J. Liu. Modularity and position independence in EDA-GP. In Proceedings of The Second Asian-Pacific Workshop on Genetic Programming, Cairns, Australia, 2004
109. Y. Shan, R. I. McKay, C. J. Lokan, and D. L. Essam. Software project effort estimation using genetic programming. In Proceedings of the IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions, volume 2. IEEE, 2002, p. 1108–1112
110. J.L. Shapiro, Drift and scaling in estimation of distribution algorithms. Evol. Comput. **13**(1), 99–123 (2005)
111. J.L. Shapiro. Diversity loss in general estimation of distribution algorithms. In Parallel Problem Solving from Nature, volume 4193 of Lecture Notes in Computer Science, Reykjavik, Iceland, September 2006. Springer, pages 92–101
112. S. Shirakawa, S. Ogino, T. Nagao, Dynamic ant programming for automatic construction of programs. IEEJ Trans. Elect. Elect. Eng. **3**(5), 540–548 (2008)
113. B. Su, Y. Shen. Maximum margin transfer learning. In Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, GEC '09, New York, NY, USA, 2009. ACM, pages 957–960
114. I. Tanev, Genetic programming incorporating biased mutation for evolution and adaptation of snakebot. Genetic Programming and Evolvable Machines **8**(1), 39–59 (2007)
115. A. Teller. Evolving programmers: The co-evolution of intelligent recombination operators. In Advances in Genetic Programming 2, chapter 3. (MIT Press, Cambridge, 1996), p. 45–68
116. A. Teller, M. Veloso. PADO: Learning tree structured algorithms for orchestration into an object recognition system. Technical Report CMU-CS-95-101, Department of Computer Science, (Carnegie Mellon University, Pittsburgh, 1995)
117. G.G. Towell, J.W. Shavlik, Extracting refined rules from knowledge-based neural networks. Mach. Learn. **13**(1), 71–101 (1993)
118. L.G. Valiant. Learning disjunctions of conjunctions. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, volume 1, Los Angeles, CA, USA, August 1985. (Morgan Kaufmann Publisher Inc., Burlington), pp. 560–566
119. C.S. Wallace, D.M. Boulton, An information measure for classification. Comput. J. **11**(2), 185–194 (1968)
120. P.A. Whigham. Grammatically-based genetic programming. In Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, p. 33–41, Tahoe City, California, USA, 9 July 1995

121. P.A. Whigham. Inductive bias and genetic programming. In Proceedings of First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, Sheffield, UK, 12-14 September 1995. IEEE, p. 461–466
122. P.A. Whigham. A schema theorem for context-free grammars. In Proceedings of the 1995 IEEE Conference on Evolutionary Computation, volume 1, Perth, Austrailia, 29 November - 1 December 1995. (IEEE Press, New York) p. 178–181
123. P.A. Whigham. Grammatical bias for evolutionary learning. PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 1996
124. P.A. Whigham, R.I. McKay, Genetic approaches to learning recursive relations. Prog. Evol. Comput. **956**, 17–27 (1995)
125. G. Wilson, M. Heywood, Introducing probabilistic adaptive mapping developmental genetic programming with redundant mappings. Genet. Program. Evolvable Mach. **8**(2), 187–220 (2007)
126. M.L. Wong and K. S. Leung. Applying logic grammars to induce sub-functions in genetic programming. In 1995 IEEE Conference on Evolutionary Computation, volume 2, Perth, Australia, 29 November - 1 December 1995. IEEE Press, pages 737–740
127. M.L. Wong, K.S. Leung, Genetic logic programming and applications. IEEE Expert **10**(5), 68–76 (1995)
128. M.L. Wong, K.S. Leung. An induction system that learns programs in different programming languages using genetic programming and logic grammars. In Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence, Herndon, Virginia, USA, 5-8 November 1995
129. K. Yanai, H. Iba. Estimation of distribution programming based on Bayesian network. In Proceedings of the 2003 IEEE Congress on Evolutionary Computation, CEC2003, Canberra, Australia, 8-12 December 2003. IEEE Press, pages 1618–1625
130. K. Yanai, H. Iba. Probabilistic model-building genetic programming based on dependency relationship. In Proceedings of The Second Asian-Pacific Workshop on Genetic Programming, Cairns, Australia, 6-7 December 2004
131. K. Yanai, H. Iba. Program evolution by integrating EDP and GP. In Proceedings of Genetic and Evolutionary Computation—GECCO-2004, Part I, volume 3102 of Lecture Notes in Computer Science, Seattle, WA, USA, 26-30 June 2004. Springer, pages 774–785
132. K. Yanai, H. Iba. Probabilistic distribution models for EDA-based GP. In GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, volume 2, Washington DC, USA, 25-29 June 2005. ACM Press, pages 1775–1776
133. M. Zlochin, M. Birattari, N. Meuleau, M. Dorigo, Model-based search for combinatorial optimization: A critical survey. Ann. Oper. Res. **131**(1), 373–395 (2004)