Genet Program Evolvable Mach (2010) 11:285–320
DOI 10.1007/s10710-010-9110-5

CONTRIBUTED ARTICLE

Theoretical results in genetic programming: the next ten years?

Riccardo Poli · Leonardo Vanneschi · William B. Langdon · Nicholas Freitag McPhee

Received: 26 November 2009 / Revised: 15 April 2010 / Published online: 15 May 2010

Abstract We consider the theoretical results in GP so far and prospective areas for the future. We begin by reviewing the state of the art in genetic programming (GP) theory including: schema theories, Markov chain models, the distribution of functionality in program search spaces, the problem of bloat, the applicability of the no-free-lunch theory to GP, and how we can estimate the difficulty of problems before actually running the system. We then look at how each of these areas might develop in the next decade, considering also new possible avenues for theory, the challenges ahead and the open issues.

Keywords Genetic programming · Theory · Challenges · Open problems

R. Poli (✉)
School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK
e-mail: rpoli@essex.ac.uk

L. Vanneschi
Department of Informatics, Systems and Communication (D.I.S.Co.), University of Milano-Bicocca, viale Sarca, 336-U14 Milan, Italy
e-mail: vanneschi@disco.unimib.it

W. B. Langdon
Department of Computer Science, King's College London, London WC2R 2LS, UK
e-mail: william.langdon@kcl.ac.uk

N. F. McPhee
Division of Science and Mathematics, University of Minnesota Morris, Morris, MN 56267, USA
e-mail: mcphee@morris.umn.edu

1 Introduction

Genetic Programming (GP) [1–5] has been remarkably successful as a problem-solving and engineering tool (e.g., see Koza's article on human-competitive results in this special issue). One might wonder how this is possible, given that GP is a non-deterministic algorithm whose behaviour varies from run to run. Why, when, and how can GP solve problems? What goes wrong when it cannot? What are the reasons for certain undesirable behaviours, such as bloat, and what can we do to avoid them without introducing new (and perhaps even less desirable) problems? These and many related questions have been posed by researchers since the early days of GP. Thus one might imagine that, by now, we ought to have a full theoretical understanding of GP's operations and a rich set of theoretically-sound guidelines for its use. However, this has only been achieved to some extent.

Despite the relative algorithmic simplicity of GP, sound theoretical models of GP and precise mathematical results have been scarce and hard to obtain, often emerging many years after the proposal of the original algorithm (e.g., see [6–10]). An important reason for this delay is that different versions of GP often require different theoretical models. The many proposed representations, such as linear, tree-based, and graph-based GP, can have very different dynamics and require different theoretical tools. Similarly, different sets of genetic operators and, in some cases, fitness/objective functions may require different theoretical models. Furthermore, the randomness, non-linearities and numerous degrees of freedom present in a typical GP system add to the already substantial challenges theoreticians are facing.

This applies not only to traditional forms of GP but also to Cartesian GP (CGP) [11, 12], Grammatical Evolution (GE) [13, 14], and Evolutionary Programming (EP) [15]. In fact, many of these challenges exist for most Evolutionary Algorithms (EAs). In the case of program induction, however, we have the additional difficulties of modelling stochastic searchers in infinite spaces, where search operators can dynamically change the dimension and structure of the solutions being explored.

So, despite recent advances in the construction of solid theoretical foundations for GP and related techniques (e.g., see [4, 9, 16] and the recent review in [5]) and the establishment of a forum where GP theoreticians and practitioners can meet and share issues and ideas (the "Genetic Programming Theory and Practice" workshop series [17]), there are still many questions that have not been fully answered, and large gaps remain between GP theory and practice.

Starting from a summary of the state of the art of GP theory (Sect. 2), we will then propose what we believe are some key open questions and promising future direction for research in Sects. 3 and 4. We conclude in Sect. 5.

2 State of the art

If we could visualise the search performed by GP, we would often find that initially the population looks like a cloud of randomly scattered points. As the generations pass this cloud changes shape and moves in the search space. Because GP is a stochastic search technique, in different runs the cloud moves in different ways. If we could see regularities, these might provide us with a deep understanding of how the algorithm is searching the program space for solutions, and perhaps help us see why GP is successful in finding solutions in certain runs and unsuccessful in others. Unfortunately, it is normally impossible to exactly visualise the dynamics of GP in a program space due to its high dimensionality and complexity.

Of course it is possible to approximately visualise the dynamics of a GP system by projecting it onto a low-dimensional space, e.g., by choosing and recording a small set of population descriptors. This is essentially what GP practitioners do all the time when plotting, for example, fitness, size and diversity statistics. There is plenty of evidence that choosing the right quantities to track may provide useful information on the dynamics of GP. However, choosing the wrong projections may obscure important details, leading researchers to formulate incorrect hypotheses.

An alternative approach to better understanding the dynamics of GP is to build and study mathematical models of evolutionary search. As we will see, there are a number of cases where this approach has been very successful in illuminating some of the fundamental processes and biases in GP systems.

In this section, we will review a number of the key tools and results, including: the use of schema theories and Markov chains (Sect. 2.2) to understand the evolutionary dynamics of GP; theoretical analysis of GP search spaces (Sect. 2.3) and what this tells us about search in those spaces; the problem of bloat (Sect. 2.4), and how theory has helped us both understand the phenomena and develop theoretically grounded approaches to controlling it (Sect. 2.4.5); problem difficulty and performance models to help us predict the behaviour of GP on unseen problems (Sect. 2.5); and the extension of the no-free-lunch (NFL) theory to GP so we can better understand what is and is not feasible (Sect. 2.6).

2.1 Schema theories

Schema theories are among the oldest and the best known models of EAs [18, 19]. Schema theories are based on the idea of partitioning the search space into subsets, called *schemata*. They are concerned with modelling and explaining the dynamics of the distribution of the population over the schemata. Modern genetic algorithm (GA) schema theory [20, 21] provides exact information about the distribution of the population at the next generation in terms of quantities measured at the current generation.

Early efforts to develop a theory of schemata in GP led to different schema theorems [1, 22–26]. However, only in the last decade have the first exact schema theories become available [27–29] that give exact formulations (rather than lower bounds) for the expected number of individuals sampling a schema at the next generation. Initially [28, 29], these exact theories were only applicable to GP with one-point crossover. However, more recently they have been extended to the class of homologous crossovers [10] and to virtually all types of crossovers that swap subtrees [8, 9], including standard GP crossover (with and without uniform selection of the crossover points), one-point crossover, context-preserving crossover and size-fair crossover, as well as more constrained forms of crossover such as strongly-typed GP crossover.

As well as being applicable to a wide variety of settings, schema-based models have generated a variety of practical results, including many of the results discussed in Sect. 2.4.

2.2 Markov chains

Markov chains are important in the theoretical analysis of EAs operating on discrete search spaces (e.g. [30]). Such algorithms are stochastic but Markovian (for example, in GAs and GP the population at the next generation typically depends only on the population at the current generation). So, all one needs to do to study their behaviour is to compute the probability that an algorithm in any particular state (e.g., with a particular population) will move to any other state (e.g., another population) at the next generation. These probabilities are stored in a stochastic matrix, M. M holds the probabilities of all the possible transitions the algorithm could make in one generation. By squaring the matrix we have the probabilities of all the transitions that can be made in two generations. M^3 contains the probabilities for three generations, and so on. By taking powers of the matrix M and multiplying the result by the initial probability distribution over states (which is given by the initialisation algorithm), it is then possible to study the behaviour of the algorithm over any number of generations. One can, in principle, compute the probability of solving a problem within n generations, the first hitting time, the average and best fitness after n generations, etc.

Markov models have been applied to GP [10, 31], but so far they have not been developed as fully as the schema theory, and have consequently generated fewer concrete results.

2.3 Search spaces

The exact mathematical models of GP seen in the previous two sections are probabilistic descriptions of the operations of selection, reproduction, crossover and mutation. They explicitly represent how these operations determine which areas of the program space will be sampled by GP, and with what probability. These models treat the fitness function as a black box, however. That is, there is no representation of the fact that in GP, unlike other evolutionary techniques, the fitness function involves the execution of computer programs on a variety of inputs. In other words, schema theories and Markov chains do not tell us how fitness is distributed in the search space. Yet, without this information, we have no way of closing the loop and fully characterising the behaviour of a GP system, since that behavior is always the result of the interaction between the fitness function and the search biases induced by the representation and genetic operations used in the system.

The characterisation of the space of computer programs explored by GP from the point of view of fitness distributions has been another main topic of theoretical research (e.g., see [4]). The main result in this area is the discovery that the distribution of functionality of non-Turing-complete programs approaches a limit as program length increases. That is, *although the number of programs of a particular length grows exponentially with length, beyond a certain threshold the fraction of programs implementing any particular functionality is effectively constant*. This happens in a variety of systems and can also be proven mathematically for LISP expressions and machine code programs without loops [4, 32–36]. The proof is based on a Markov chain model of program execution. Its key ideas are sketched below.

To gather information about the space of programs, one can start from a particular program, and modify it using appropriate operators so as to obtain other programs. If this is done in an exhaustive way we can gather information about the whole space of programs. One can do that by building programs by progressively adding randomly chosen instructions to them and evaluating how their functionality changes as more and more instructions are added. In other words, we can interpret a program containing ℓ instructions as a stochastic vector $v = (v_1 \ldots v_\ell)$ where each v_i represents the choice of the i-th instruction in the program, and we can construct a probability tree where the i-th level represents all possible outcomes of adding one instruction to all possible programs of length $i - 1$. The edges in the tree represent the probability of choosing each element of the instruction set to fill the i-th position in a program.

It is clear that this probability tree is infinite. However, a digital computer with bounded memory can only be in one of a finite number of states at any given time, and a program can only transform the original state of the machine to one other. Therefore, there are only a finite number of state transformations (functions from states to states) that programs can implement. It follows that each node in the search/probability tree of programs must correspond to one of a finite set of state transformations. So, the search/probability tree induces a *finite* graph, the nodes of which represent state transformations. There is an edge between two nodes in the graph if it is possible to transform the "functionality" of one state transformation (source node) into another (destination node) by executing one of the instructions after performing the first transformation. The edges are weighted by probabilities that are computed from the weights in the probability tree. If we draw instructions randomly and independently when extending programs, the weights in the probability tree are identical at each level in the tree. As a result the graph can be interpreted as the transition matrix of a Markov chain. The chain describes how the state transformation performed by programs changes as we add new instructions.

The iteration of the chain corresponds to exploring the state-transition behaviour of progressively longer programs. In particular, we can exactly compute how the probability distribution of behaviours changes as a function of program length, ℓ, namely $p = M^\ell\,(1\ 0 \ldots 0\ )^T$, where we assumed the identify function is the first state of the chain. Under mild conditions, one can prove that taking the limit as $\ell \to \infty$, the distribution of functionality, p, for a space of programs approaches a limit.

That the limiting distribution of functionality reaches a limit as program length increases has also been proven for a variety of other non-Turing-complete computers and languages [5, 34]. Also, [37] started extending these results to Turing complete programs written in a simple, but realistic, machine code language, T7. Effectively this work recasts the classical undecidable "Halting Problem" into a probabilistic question: What is the probability that a program chosen at random will halt? The analysis indicated that the *halting probability* for T7 programs of length ℓ is of order $1/\sqrt{\ell}$. Experimental results confirmed the theory. Some theoretical properties of Turing-complete GP have also been explored in [38].

Recently, several ideas have been put forward on how to analyse or improve search in program spaces by explicitly representing the semantics/functionality of programs (in addition to or instead of their syntax). For example, a mechanism for studying the impact of subtree crossover in terms of semantic building blocks was presented in [39], while the possibility of never resampling the same functionality during a search in program spaces thanks to a canonical representation for programs was suggested in [40].

2.4 Bloat

It is extremely common for the average number of nodes of the programs in a GP population to remain largely static for several generations, after which it starts growing rapidly. Typically the increase in program size is not accompanied by any corresponding increase in fitness. This phenomenon is known as *bloat*. Its origin has been the focus of intense research for well over a decade. Bloat is not only interesting from a scientific point of view, but has significant practical deleterious effects, slowing down the evaluation of individuals, consuming large computational resources, and making solutions hard to interpret.

Note that there are situations where one would expect to see program growth as part of the process of solving a problem. For example, in continuous symbolic regression problems, GP runs typically start from populations of small random programs. However, it may be necessary for the programs to become larger for them to be able to fit all the test cases. So, bloat is typically defined as *program growth without (significant) return in terms of fitness*. In other words, growth is only a necessary condition for bloat.

Bloat was one of the earliest GP phenomena to be studied theoretically, and over the years theory has yielded both increasing understanding of bloat and powerful, theoretically grounded tools for combating it.

2.4.1 Three classic explanations for bloat

We start with three early theories of bloat. While each of these provided valuable intuition, none provided an explanation that could be used quantitatively, making it difficult to use them as a basis for controlling bloat.

The oldest theory (perhaps first suggested by Singleton [41]) is that bloat is evolved by GP, as *inactive code* (code that is not executed, or is executed but its output is then discarded) increases, to protect useful code from the effects of crossover. Fit individuals with more inactive code are less likely to be disrupted by crossover, so their children are more likely to be as fit as they are. Thus more of their children survive to themselves have children. Since the protective advantage of being larger is both inherited and increases with size, there is a continuous tendency to increase in size [3, 42–44]. This theory is known as the *protection against crossover* or the *replication accuracy* theory.

The *removal bias theory* [45] observes that inactive code in a GP tree tends to be low in the tree, thus residing in smaller-than-average-size subtrees. Crossover events excising inactive subtrees produce offspring with the same fitness as their parents. On average the inserted subtree is bigger than the excised one, so such offspring are bigger than average while retaining the fitness of their parent, leading ultimately to growth in the average program size.

Finally, the *nature of program search spaces theory* [46, 47] predicts that above a certain size, the distribution of fitnesses does not vary with size. Since there are more long programs, the number of long programs of a given fitness is greater than the number of short programs of the same fitness. Over time GP samples longer and longer programs simply because there are more of them.

2.4.2 Executable models of bloat

The explanations for bloat provided by these three theories are largely qualitative. There have, however, been some efforts to mathematically formalise and verify them. For example, [48] defined an *executable model of bloat* where only the fitness, the size of active code and the size of inactive code were represented. Fitnesses of individuals were drawn from a bell-shaped distribution, while active and inactive code lengths were modified by a size-unbiased mutation operator. Rosca [49] proposed a slightly more sophisticated model that also included an analogue of crossover.

Executable models are appealing because of their simplicity. However, since they omit many details of the representation and operators typically used in GP, it is difficult to verify if all the phenomena observed in the model have analogues in GP runs, and if all important behaviours in relation to bloat are captured by the model.

2.4.3 Size evolution equation

Starting from the exact schema theory, in [9, 50] a *size evolution equation* for GP was developed, which provided an exact formalisation of the dynamics of average program size. The equation has recently been simplified [51] as

$$E[\mu(t+1)] = \sum_{\ell} \ell \times p(\ell, t), \tag{1}$$

where $\mu(t + 1)$ is the mean size of the programs in the population at generation $t + 1$, E is the expectation operator, ℓ is a program size, and $p(\ell, t)$ is the probability of selecting programs of size ℓ from the population in generation t.

Equation (1) can be written in terms of the expected change in average program size as

$$E[\mu(t+1) - \mu(t)] = \sum_{\ell} \ell \times (p(\ell, t) - \Phi(\ell, t)), \tag{2}$$

where $\Phi(\ell, t)$ is the proportion of programs of size ℓ in the current generation. Both equations apply to a GP system with selection and any form of symmetric subtree crossover. (In a symmetric operator the probability of selecting particular crossover points in the parents does not depend on the order in which the parents are drawn from the population).

As shown in [51], for fitness proportionate selection, Eq. (2) is equivalent to applying Price's covariance selection theorem [52] to program lengths. The use of Price's theorem to predict the evolution of size had been advocated over a decade ago in [53] (also reported and expanded in [4]). While the behaviour of genetic operators depends on the arity of the primitives, which could potentially invalidate the applicability of Price's theorem, it was argued in [4] that this dependency is unlikely to play any major role in the evolutionary dynamics under normal conditions. The independent derivation of Price's theorem from the schema theory in [51] essentially provides a formal proof of this conjecture, showing that Price's result does indeed hold for symmetric crossovers.

Note that Eqs. (1) and (2) do not directly explain bloat. They are, however, important because *they constrain what can and cannot happen size-wise* in GP populations. In particular, they predict that, for symmetric subtree-swapping crossover operators, the mean program size evolves *as if selection only was acting* on the population. Thus, if there is a change in mean size (bloat, for example) it must be the result of some form of positive or negative selective pressure on some or all of the length classes ℓ. More specifically, there can be bloat only if the selection probability $p(\ell, t)$ is different from the proportion $\Phi(\ell, t)$ for at least some ℓ. Any explanation for bloat has to agree with these constraints.

2.4.4 Crossover bias theory of bloat

We conclude this review of theories of bloat with a recent explanation for bloat called the *crossover bias theory* [54, 55]. This is based on, and is consistent with, the size evolution equation (Eq. 1).

On average, each application of subtree crossover removes as much genetic material as it inserts; consequently crossover on its own does not produce growth or shrinkage. While the *mean* program size is unaffected, however, *higher moments* of the distribution are. More specifically, crossover pushes the population towards a particular family of distributions of program sizes, known as a *Lagrange distributions of the second and third kind* [54–56], where small programs have a much higher frequency than longer ones. For example, crossover generates a very high proportion of single-node individuals. In virtually all applications, very small programs have no chance of solving the problem. As a result, programs of above average size have a selective advantage over programs of below average size, and the mean program size increases.

Because crossover will continue to create small programs, which will then be ignored by selection (in favour of the larger programs), the increase in average size will continue generation by generation.

2.4.5 Theoretically-sound bloat control

Numerous techniques have been proposed to control bloat [47, 57]. These include the use of size and depth limits, the use of genetic operators with an inbuilt anti-bloat bias [58], the use of multi-objective selection techniques to jointly optimise fitness and minimise program size [53], and, of course, the famous parsimony pressure method [1, 59]. Most of these have been largely ad hoc, but several bloat control techniques have been proposed that are inspired by and/or based on the findings of GP theory. We briefly review a few of these techniques in this section.

As clarified by the size evolution equation (see Sect. 2.4.3), in systems with symmetric operators, bloat can only happen if there is an imbalance between selection probabilities and frequencies of programs, e.g., if some longer-than-average programs are fitter than average or some shorter-than-average programs are less fit than average, or both. So, it stands to reason that *in order to control bloat one needs to somehow modulate the selection probabilities of programs based on their size.*

The well-known *parsimony pressure* method [1, 59, 60, 61] changes the selection probabilities by subtracting a value based on the size of each program from its fitness. Bigger programs have more subtracted and, so, have lower fitness and tend to have fewer children.

That is, the new fitness function is $f(x) - c \times \ell(x)$, where $\ell(x)$ is the size of program x, $f(x)$ is its original fitness and c is a constant known as the *parsimony coefficient*. (While the new fitness is used to guide evolution, one still needs to use the original fitness function to recognise solutions and stop runs).

Zhang [59] showed some benefits of adaptively adjusting the coefficient c at each generation, but most implementations still use a constant parsimony coefficient. Since the intensity with which bloat is controlled is determined by the parsimony coefficient, its value is very important: too small a value and runs will still bloat wildly; too large a value and GP will take the minimisation of size as its main target and will almost ignore fitness, thus converging towards extremely small but useless programs [62]. However, good values of the parsimony coefficient are highly dependent on particulars such as the problem being solved, the choice of functions and terminals, and various parameter settings. Furthermore, since the pressure to bloat changes over time, a constant parsimony coefficient can only achieve *partial control* over the dynamics of the average program size over time.

Recently, a method for setting the parsimony coefficient in a principled manner has been proposed [63]. The method, called *covariant parsimony pressure*, was developed by feeding a parsimonious fitness function within the "Price's theorem version" of Eq. (2). The method is very easy to implement. It computes the parsimony coefficient c as $c = \mathrm{Cov}(\ell, f) / \mathrm{Var}(\ell)$, where $\mathrm{Cov}(\ell, f)$ is the covariance between program size, ℓ, and program fitness, f, in the population, and $\mathrm{Var}(\ell)$ is the variance of program sizes. Note that c needs to be recomputed at each generation because both $\mathrm{Cov}(\ell, f)$ and $\mathrm{Var}(\ell)$ change from generation to generation. Using this equation ensures that the mean program size remains at the value set by the initialisation procedure. There is a variant of the method that allows the user to even decide what function the mean program size should follow over time thereby providing complete control over the population size dynamics.

The *Tarpeian method* [64], controls bloat by acting directly on the selection probabilities in Eq. (2). This is done by setting the fitness of randomly chosen longer-than-average programs to 0, preventing them from being parents. An advantage of the method is that the programs whose fitness is zeroed are never executed, thereby speeding up runs. The anti-bloat intensity of Tarpeian control can be modulated by changing how frequently large individuals are penalised. As recently shown in [65] this can be done dynamically to gain complete control over the dynamics of mean program sizes.

Finally, the realisation of the importance of the precise characteristics (not just the mean) of the size distribution of program sizes resulting from the studies on the biases of crossover has lead to another theoretically-inspired method to counteract bloat: *operator equalisation* [66]. By filtering which individuals are accepted in each new generation, this technique allows accurate control of the distribution of program lengths inside the population, allowing a user to obtain virtually any desired distribution. Interesting variations of this technique have been proposed in [67, 68].

2.5 Problem difficulty and performance models in GP

What problems are suitable for GP? Is it possible to measure how hard or easy a problem is for GP? In classical algorithms we have a well-developed complexity theory that allows us to classify problems into complexity classes such that problems in the same class have roughly the same complexity, i.e., they consume (asymptotically) the same amount of computational resources, usually time. Although, properly speaking, GP is a randomised heuristic and not an algorithm, it would be nice if we were able to likewise somehow classify problems according to some measure of *difficulty*.

Difficulty studies have been pioneered in the field of GAs, by Goldberg and coworkers (e.g., see [69, 70]). Their approach consisted in constructing functions that should a priori be easy or hard for GAs to solve. These ideas have been followed by many others (e.g. [71, 72]) and have been at least partly successful in the sense that they have been the source of many proposals as to what makes a problem easy or difficult for GAs. One concept that underlies many of these approaches is the notion of *fitness landscape*.

The metaphor of a fitness landscape [73], although not without faults, has been a fruitful one in several fields and is attractive due to its intuitiveness and simplicity. In EAs a fitness landscape is a plot where the horizontal axes represent the different individual genotypes in a search space and the vertical axis represents the fitness of each of these individuals [4]. If genotypes can be visualised in two dimensions, the plot can be seen as a three-dimensional "map", which may contain peaks and valleys. For maximisation problems, finding the best solution is equivalent to finding the highest peak.

One early example of the application of the concept of fitness landscape to GP is represented by the work of Kinnear [74], where GP difficulty was related to the shape of the fitness landscape and analysed through the use of the fitness *autocorrelation function*. The work of Nikolaev and Slavov [75] represents the first effort to apply to GP a difficulty measure called *fitness distance correlation* (FDC) (first studied by Jones [76] for GAs), with the aim of determining which mutation operator, among a few that they propose, "sees" a smoother landscape on a given problem. The pros and cons of FDC as a general difficulty measure in GP are investigated in [77, 78]. While FDC can give reliable predictions, one of its major drawbacks is that the genotype of the global optimum must be known beforehand in order to calculate it.

In order to overcome this limitation, a new empirical hardness measure based on the analysis of offspring-fitness versus parent-fitness scatterplots, called *negative slope coefficient* (NSC), has been recently presented in [79, 80] where its usefulness for GP has been investigated. Results indicate that, although NSC, too, is not without faults, it is a suitable hardness indicator for many well-known GP benchmarks, for some hand-tailored theoretical GP problems and also for some real-life applications [81].

While both FDC and NSC have generated useful results, both fail to take into account many typical characteristics of GP. For instance, so far the NSC definition only includes (various kinds of) mutation, but does not include crossover. A similarity/dissimilarity measure related to standard sub-tree crossover has been presented in [82–84]. Hopefully, in the future this will allow the definition of more representative measures of problem difficulty for GP. This subject represents an important open issue in GP and is discussed in more details in another article in this special issue [85].

In addition, while FDC and NSC (as well as other indicators of problem difficulty) have been shown to provide reasonably reliable measures of *whether a problem is hard or easy*, they do not really give a direct estimation of *how hard or easy* a problem is in relation to any particular performance measure. That is, they cannot easily be used to estimate, for example, the success rate of a GP system or the expected number of fitness evaluations required to solve a problem. Recent work [86, 87] has begun to rectify this situation. The work has found that a simple semi-linear model can accurately predict the performance of program-induction algorithms for the domains of symbolic regression and Boolean function induction. The model computes the distance between a set of reference problems and the problem for which we want to estimate performance. It predicts performance via a linear combination of such distances. The model's coefficients and the particular problem instances to be used as the reference set are automatically instantiated through the use of a training set of problems.

Surprisingly, these simple models were able to accurately predict the performance of GP on unseen problems. The models have also been used in the analysis and comparison of different algorithms and/or algorithms with different parameter settings and for the automatic construction of algorithm taxonomies (see Sect. 4.2).

2.6 No-free lunch and GP

Informally speaking, the no-free-lunch (NFL) theory originally proposed by Wolpert and Macready [88] states that, when evaluated over all possible problems, all algorithms are equally good or bad irrespective of our evaluation criteria. In the last decade there have been a variety of results that have refined and specialised NFL (see [89] for a comprehensive recent review). One such result states that if one selects a set of fitness functions that are closed under permutation then the expected performance of any search algorithm over that set of problems is constant, i.e., it does not depend on the algorithm we choose [90]. The result is valid for any performance measure. Furthermore, [90] showed that it is also the case that two arbitrary algorithms will have identical performance over a set of functions only if that set of functions is closed under permutation.

What does it mean for a set of functions to be closed under permutation? A fitness function is an assignment of fitness values to the elements of the search space. A permutation of a fitness function is simply a rearrangement or reshuffling of the fitness values originally allocated to the objects in the search space. A set of problems/fitness functions is closed under permutation, if, for every function in the set, all possible shuffles of that function are also in the set.

Among the many extensions of NFL to a variety of domains, Woodward and Neil [91] argued that there is a free lunch (a cumulative advantage for some search algorithms) in search spaces where there is a non-uniform many-to-one genotype-phenotype mapping. Since the mapping from syntax to functionality (or semantics) in GP is one such mapping (e.g., see [92]), we might expect a free lunch for GP. The reason why NFL would not normally be applicable to search in program spaces is that there are many more programs than functionalities and that not all functionalities are equally likely. So, interpreting syntax trees as genotypes and functionalities as phenotypes, the GP genotype-phenotype mapping is many-to-one and non-uniform, which invalidates NFL.

In recent work [93] it has been possible to extend these results to the case of standard GP problems and systems and to ascertain that a many-to-one genotype-phenotype mapping is not the only condition under which NFL breaks down when searching program spaces. The theory is particularly easy to understand because it is based on geometry. We briefly review it in the remainder of this section.

If, for simplicity, we concentrate on programs with a single output, the fitness of a program p in GP is often the result of evaluating the difference between p and a target behaviour over a number of fitness cases, i.e., $f(p) = \sum_{i=1}^{n} |p(x_i) - t_i|$, where f is the fitness function, $\{x_i\}$ is a set of fitness cases of cardinality n, $p(x_i)$ is the evaluation of individual p on fitness case x_i, and t_i is the target value on fitness case x_i. Note that if the set of fitness cases $\{x_i\}$ is fixed, the fitness function is determined by the vector of target values $\mathbf{t} = (t_1, t_2,\ldots, t_n)$ and the behaviour of program p is fully represented by the vector $\mathbf{p} = (p(x_1), p(x_2), \ldots, p(x_n))$. Note also that a sum of absolute differences is a *city-block distance*. Thus, we can see the fitness assigned to a program as the distance, $d(\mathbf{p}, \mathbf{t})$, between the vectors $\mathbf{t}$ and $\mathbf{p}$. In other words, if we know the fitness f of a program, we know that the target behaviour $\mathbf{t}$ that generated that *fitness must be on the surface of a city-block sphere* centred on $\mathbf{p}$ having radius f.

Using this, [93] showed that, if $\{\mathbf{p}_1, \mathbf{p}_2,\ldots, \mathbf{p}_r\}$ is a program search space, $\mathcal{F} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_m\}$ is a set of fitness functions of the form $f(\mathbf{p}) = d(\mathbf{p}, \mathbf{t})$, and $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_m\}$ is the set of target vectors corresponding to such functions, the set $\mathcal{F}$ is closed under permutation (and NFL applies to it) *if and only if* a particular

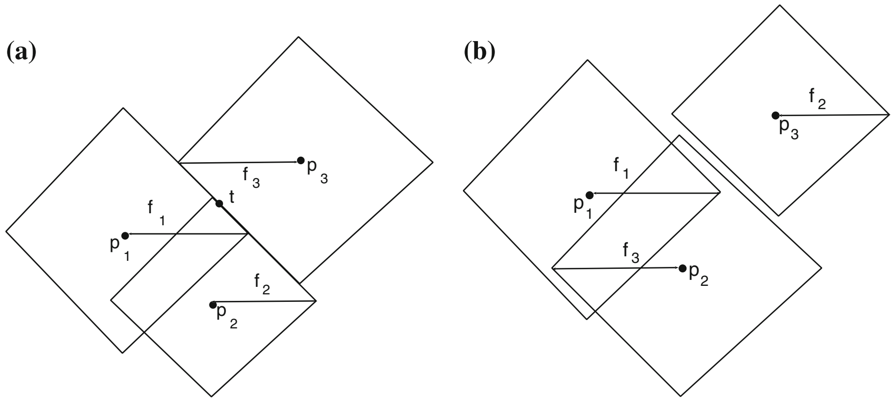

Fig. 1 Geometric representation of one valid (**a**) and one invalid (**b**) fitness function. In this simple example we assume that the search space contains only three programs (hence the three city-block spheres). The fitness function in (**b**) has been obtained via a permutation of the function in (**a**)

set of geometric constraints are satisfied. More specifically, the target vector $\mathbf{t} \in \mathcal{T}$ associated to a function $\mathbf{f} \in \mathcal{F}$ must be at the intersection of the city-block spheres centred on programs $\mathbf{p}_1$, $\mathbf{p}_2,\ldots$, $\mathbf{p}_r$ and having radii $f_1, f_2,\ldots, f_r$, respectively, as illustrated in Fig. 1a. Permuting the elements of $\mathbf{f}$ to obtain a new fitness function corresponds to shuffling the radii of the spheres centred on $\mathbf{p}_1$, $\mathbf{p}_2,\ldots$, $\mathbf{p}_r$. After the shuffling some spheres may have had their radius decreased, some increased, and some unchanged. If any radius has changed then $\mathbf{t}$ can no longer be the intersection of the spheres. However, we must be able to find a new intersection or else the new fitness function we have generated is invalid (like the one shown in Fig. 1b) and, thus, cannot be a member of $\mathcal{F}$.

With this result in hand, [93] identified easily-satisfied conditions where there is a free lunch for GP. Many of these are based on geometry (e.g., derived from the triangular inequality and the inverse triangular inequality). In general, it is extremely easy to find realistic situations in which a set of GP problems is provably not closed under permutation. This implies that we can expect that there is a free lunch for techniques that sample the space of computer programs. This is particularly important since it implies that, for GP, it is worthwhile to try to come up with new and more powerful algorithms.

3 The next decade: extending previous avenues

Despite the significant theoretical steps described in the previous section, a number of major issues remain open; in this and the next section we outline some of these. We hope that this overview will help to focus future research in order to both deepen our understanding of GP and allow the development of more powerful program induction algorithms. We start with ideas for extending the avenues described in the previous section.

3.1 Schema theories

In principle, exact GP schema theories allow one to predict the state of the population in the next generation based on the information we have on its state in the present generation. The prediction is probabilistic, in the sense that we can compute expected frequencies of programs, we can compute their variances, or even the full probability distribution over populations. Note, however, that the minimum requirement to be able to predict the next generation is to work with the schema equations for *all* possible programs that could be generated from that population, irrespective of how unlikely their creation might be. Thus, for typical populations the number of schema equations to keep track of is immense. Furthermore, if one wants to study what happens with all possible populations, and we do not artificially limit the search space to be finite, then we have the problem of handling infinitely many equations. Also, while each schema equation is conceptually not too complex (it is a weighted sum of products of selection probabilities), the number of terms it includes may be phenomenally large as it grows proportionally to the square of the number of program shapes times the square of the number of possible crossover points.

Another key problem is the fact that schema equations predict the state of the population in the next generation, but not two or more generations in the future. This longer-term dynamics, however, is what normally interests researchers the most. E.g., we would like to know how likely a GP system is to find a solution 50 generations in the future.

Because of these obstacles, it has only been possible to use schema equations to predict the dynamics of GP in cases where they naturally simplify. For example, if one assumes that only one operator is acting on the population (e.g., only selection, only mutation or only crossover) and if one further assumes that the population is large enough that we can ignore stochastic effects, then it is possible to integrate the schema equations over multiple generations and find their fixed points. This approach has already provided useful information about the search biases of crossover and mutation. However, it is hard to predict the dynamics of GP in more realistic situations, such as when crossover and selection are both used.

Schema equations are a form of *coarse graining* or averaging of the dynamics of GP, where we replace the original degrees of freedom of a system with new quantities that represent the collective state of many such degrees of freedom. A question, then, is whether coarse graining the dynamics of GP using schemata is a good way to proceed. Recent work has shown that schema equations are the result of rewriting the dynamics of GP in a new reference system, called the *building block basis* [16]. The building block basis for GP appears to be excellent at capturing the regularities present in homologous crossover operators and represents most obviously the appropriate effective degrees of freedom in the case of weak selection and strong crossover. The question remains though, which coarse graining and, more generally, which basis is most appropriate for a different type of GP system, particularly one where sub-tree crossover is used and selection provides strong guidance to evolution. We currently only have answers for a small set of special cases. It is likely that the answer will depend not only on the specific features of the algorithm we want to model but also on the particular problem or class of problems we intend to solve. Also, to properly evaluate alternative bases one would really need to define some quality measure that can be used to assess the degree to which a particular coarse graining simplifies the dynamics of the GP system. A more general understanding of the different bases themselves is, we believe, also of great importance. The building block basis for GP is, for example, still a fairly recent development and much remains to be understood about it.

Another major challenge for schema theories is to move beyond trees and linear structures to a world of more general graphs. Graphs are perhaps the most powerful representation available in computing. Anything from linear structures, to parallel systems, to neural networks, to organisations, etc. can be represented with graphs of one type or another. Extending the GP schema theory to this type of structures would give it an enormous scope, but so far only a tiny amount of progress has been made, e.g., [94].

Currently, exact schema-based models only exist for homologous crossover operators and subtree-swapping style operators. However, these two classes of operators are two extremes of a continuum: one represents the case in which perfect alignment of structure is imposed on the trees (or linear structures) undergoing crossover, the other represents the case in which alignment between the parents is not even attempted. However, this continuum is full of interesting alternatives. For example, in nature, the alignment of DNA strands is based on a matching process between bases (and, consequently, between genes). It would be interesting to extend this notion to linear-GP chromosomes and tree-like structures and be able to theoretically model this type of process. Generally, being able to categorise, characterise and model the operators in the continuum between homologous and subtree crossovers is an important challenge. The same extension should be undertaken for unary operators, where GP schema theory is currently limited to subtree-type of mutations and point-mutation.

Although the development of exact evolution equations, such as the exact GP schema theory, dates back several years there remains a disquieting lack of tools with which solutions to the equations may be found. In particular, we know of no systematic approximation schemes that have been studied, though several come to mind, such as an expansion around the strong selection limit. Alternatively, in the strong-crossover, weak-selection limit an expansion in principle should be possible in terms of the deviation from a perfectly flat landscape. Often, expansions require an exact solution around which to expand. Hence, it is important to find as many exact solutions as possible that may serve as starting points of an expansion, and much of this remains to be done.

3.2 Markov chains

As we indicated in Sect. 2.2, there has been initial work in extending Markov chain models of population dynamics to GP. In [10] this was achieved for a GP system based on homologous crossover by passing through the schema equations for that system. Since exact schema equations are also available for GP with ordinary sub-tree crossover [8, 9] in principle one could derive a Markov chain model for standard GP.

Markov chain models of evolutionary algorithms are typically immense. Essentially this is because EAs, and GP in particular, have numerous degrees of freedom, and a Markov chain must keep track of them all. This makes Markov chain models explode exponentially. If the GP search space is not limited, one would in fact be confronted with infinitely large transition matrices for the chain, with the obvious difficulties that this would entail.

This is why in other areas of EAs, Markov chain models are rarely explicitly written down or even numerically integrated. Instead, researchers are interested in understanding the properties of the Markov transition matrix associated with a system. For example, the study of the eigenvalues and eigenvectors of the transition matrix can reveal important properties of the dynamics of a system.

This still needs to be done for the Markov model of GP with homologous crossover proposed in [10] and would need to be done for future Markov models of GP with other operators.

It would be interesting to use Markov analysis to see if there are problems and/or GP parameter settings where the dynamics are essentially the same. If so, we might expect phase transitions in the dynamics at the boundaries between groups of similar problems or areas of the parameter space. Perhaps studying the transition matrix will allow us to map some or all of these phase transitions, in the way that physics has mapped how the phase transition of water into steam varies with pressure and temperature. Another interesting parameter is the hitting time for specific states. If one knew that a GP system is ergodic, one could then try to estimate the expected number of generations (or fitness evaluations) before a solution is first visited. It might be possible, in this respect, to apply the ideas of [95] where bounds for the first hitting time of some EAs were derived.

3.3 Search spaces

As we have seen in Sect 2.3, some steps have been made towards understanding the characteristics of program spaces. However, much remains to be discovered.

We now understand what happens when programs grow sufficiently large, i.e., the distribution of functionality reaches a limit. However, since GP runs normally start with relatively small programs, a lot of the action happens before this limit is reached. For example, it is clear that the quality of short programs may have a significant effect on bloat. Also, for specific problems (e.g. [96]) we know that there are sweet spots in program length where finding solutions is easier than elsewhere. It would, therefore, be enormously important to obtain theoretical models that can explain where the fitness sweet spots and fitness ditches are and why. Also, knowing that there is a limiting distribution of functionality is important but knowing what that distribution is going to be would be at least equally important. Yet, we know very little about this.

Also, theoretical models of Turing-complete GP are, so far, only available for tiny languages, and these models are approximate. In the future it would be important to obtain more general and precise models than these. In particular, it seems feasible to extend the Markov chain approach used to study non-Turing-complete program spaces to the case of Turing complete systems (with finite memory). This might be possible if program memory and the program counter register are considered as part of the state of the Markov process modelling computation. This might give us new Markov chains from which properties of programs could be inferred, including whether and in which conditions they terminate. It might then be possible to study the halting problem from a radically different perspective than the classical logic-based approach.

3.4 Search biases and bloat

As we have seen in Sect. 2.4.4, the search bias of genetic operators and its interaction with the bias of selection may have important consequences on how GP behaves. The dynamics of the population is always a compromise between all such biases. Selection focuses the search towards the better individuals in the population. Crossover and mutation produce variations that typically go against the pressure of selection. Although selection is well understood, we do not yet have a complete understanding of the biases of the search operators.

The main focus of research so far has been on biases related to the size of programs. We know, for example, that subtree crossover pushes the population towards Lagrange distributions [54, 56]. However, after some early efforts that mainly focused on linear structures [7, 97–99], the study of the size biases of subtree mutation appears to have been abandoned prematurely with very few exceptions (e.g., [100]).

Size, however, is not the only element that matters. Biases related to the distribution of primitives within programs are also extremely important. For many GP systems and problems, in order to achieve the target functionality a GP system needs to find a program that presents the right primitives in the right order. So, selection is likely to increase the correlation between loci within the representation. By their own nature, search operators need to reduce correlations in order to generate novel individuals and thereby explore program space. Any changes to primitives or their order is likely to break the functionality of a program. So, precisely what type of correlations get reduced and in what ways is extremely important. If an operator respects certain type of structures, unavoidably programs having many such structures will reproduce better (will be more evolvable) than programs carrying more “fragile” structures.

While we have a relatively clear picture of the biases of homologous GP crossovers [31, 101] in relation to primitive frequencies, we have only started scratching the surface for subtree crossover. In particular [102, 103] found that there are migrations of primitives (a diffusive bias) within the representation. Other studies have found repetitive structures both in linear representations and trees [104, 105]. We need to characterise these and understand in what ways they interfere with or contribute to the functioning of a GP system.

It is possible, in fact, that these hidden migrations of primitives within the representation are just as important to the dynamics of GP as bloat. Research to date has concentrated so much on program size and bloat because the effects of the size biases of crossover are apparent to anyone running a GP system. This does not mean, however, that less apparent biases, such as the diffusive bias of subtree crossover, are less important and need no countermeasure. The opposite might well be true.

An emerging area in the analysis of the biases of GP operators is the study of their geometric properties [100, 106]. It is, indeed, possible to formally interpret the action of some tree-based GP operators as a move in a geometric space endowed with a distance. This reveals that children produced by crossover are elements of the line segment connecting parent programs while children produced by mutation are random elements from a sphere centred on a parent program. This line of research has recently made it possible to formally extend particle swarm optimisation and differential evolution to the exploration of program spaces [107, 108].

New studies of the biases of search operators (discussed in this section) and of the properties of program search spaces (discussed in Sect. 3.3) are likely to give us a more complete understanding of bloat. At present, we certainly have very strong evidence that the overproduction of short programs caused by the intrinsic bias of subtree crossover coupled with the relatively poor fitness of such programs is an important (perhaps even the most important) cause of bloat in many cases. However, it is not clear to what extent other elements (such as the removal bias) can contribute to the problem. It would be desirable if future research could address this.

3.5 Problem difficulty and performance prediction

Although much of what fitness distance correlation could offer has been understood (see for instance [77]), some important improvements are possible and deserve to be investigated. First of all, FDC could be calculated on a sample generated with more sophisticated techniques than the ones investigated in [77] (uniform random sampling or methods of importance sampling like Metropolis or Metropolis-Hastings). The use of those techniques could hopefully lead to the definition of an FDC measure that could be effective on functions like the ridged ones, where only a small portion of the search space really matters. Moreover, FDC could be used to test the suitability of distance metrics: a new research track has recently started [82–84] aimed at defining a dissimilarity measure for tree structures that should be coherent with standard GP crossover. Such a measure could be very useful to more deeply understand the dynamics of standard GP. For example it could be used for more precise calculations of genotypic diversity, or to investigate the pros and cons of standard GP crossover for different classes of problems. Testing the efficacy of FDC, calculated with such a dissimilarity measure could be a good way to validate its suitability and effective relationship with the genetic operators. Finally, correlation is not necessarily the best way to quantify the relationship between fitness and distance to the goal. Other statistics, e.g., rank-based, deserve to be investigated.

For fitness clouds in general, and the negative slope coefficient in particular, the situation is almost opposite than for FDC. Only little of what this approach can offer has been achieved [77–80]. There are many different ways of analysing fitness clouds in relation to evolvability and problem difficulty. For instance, more sophisticated and formal mechanisms might better split the fitness cloud into bins. Other sampling techniques can be investigated for generating the fitness cloud. These might reveal new and unexpected features of fitness landscapes. The number of neighbours to be considered for each sampled point, the way of choosing them and the genetic operators to be used to generate neighbourhoods are elements that also deserve a deeper investigation. Furthermore, NSC results have not been normalised so as to fall into a fixed range of values and, thus, they cannot be compared across different problems. For this reason, so far NSC has not been used to classify GP problems on the basis of their hardness, which, thus, remains an interesting and stimulating open challenge. Furthermore, analysing the slope of segments joining bins centroids is not the only way of characterising fitness clouds: for instance, their shape and their position probably contain information that may potentially lead to the definition of new algebraic measures of problem difficulty. Finally, it might be possible to go deeper that then level of fitness when studying evolvability in GP. For example, we could characterise the relation between parent and child programs via their behaviour in the fitness cases.

Naturally FDC and fitness clouds are not the only ways of studying GP problem difficulty. For example, as suggested in [page 110,4], schema theory could be an effective tool for doing this. In the future, this might allow us to identify rigorous strategies to calculate the computational effort required to solve a given problem using GP. Markov chain models could equally allow this. The availability of rigorous computational effort equations would pave the way to a precise definition of easy fitness functions for a particular form of GP. Such functions could simply be those for which the number of fitness evaluations necessary to find a solution with say 99% probability in multiple runs is (much) smaller than 99% of the effort required by exhaustive search or random search without resampling. Hard fitness functions could symmetrically be defined. More generally, having a theory to compute (or estimate) the computational effort for a problem would allow one to study what effects changing the GP system would have on the difficulty of that problem. For example, one could vary population size, the selection pressure, the genetic operator rates, the size limits for individuals, and so on. It would also be possible to study the difficulty of entire classes of problems. McPhee and Poli [109] made some first steps in this direction for GP, and [110] proposed a strategy for GAs that might allow one to predict the probability of obtaining a solution for a problem in a fixed number of generations using the schema theory.

Whether existing formalisms will prove adequate for the task of modelling problem difficulty is an open question. It is possible that the dynamics of GP will need to be expressed using new, yet to be discovered, degrees of freedom (most likely coarse grained ones) to do that.

As we have seen in Sect. 2.5, the simple performance modelling techniques for GP proposed in [86, 87] offer an alternative way of assessing performance, and, hence, difficulty for GP. While these models are simple and approximate, they have provided accurate predictions. The predictions also allow the comparison of algorithms with different parameter settings, reproduction strategies, etc. In recent work [111], these models have also been used to solve the algorithm selection problem (i.e., the problem of deciding which is the best GP algorithm to solve a problem). It appears possible to extend these models to look at how population size and other parameters of a system influence performance. Also, it would be important to explore whether there are some performance measures and problem classes for which the approach is particularly suitable or unsuitable and why.

Finally, as highlighted by one of the reviewers of this paper, we should note that while complexity theory has not been applied to the problems that GP methods face, it is likely that, in the next decade, we will see some work in this area that will allow us to compare and classify GP problem difficulty. For example, there is a great deal of work in the traditional computer science community on complexity analysis of graph-based problems; if someone could find a reduction of a particular GP problem to a common graph problem, or borrow analysis methods, one might be able to determine explicit general complexity bounds for a GP problem (independent of any specific GP method).

3.6 Free lunches

As we have seen in Sect. 2.6, there can be a free lunch for techniques that sample the space of computer programs. The non-applicability of no-free-lunch (NFL) means that it may be worthwhile trying to come up with better algorithms for function or program induction.

Naturally, the fact that NFL results do not hold for these problem domains does not imply that the existing methods for searching program and function spaces, such as the many variants of GP, are average, better than average or worse than average. So, future research should attempt to prove where such algorithms reside in a "free lunch" scale. Also, future research should try to determine when it is possible to distinguish between program induction algorithms and which performance measures allow differences.

Finally, NFL really tells us something about the mean performance of an algorithm. However, in really difficult problem domains (and automatic programming is likely to be one of them) the mean performance of algorithms may not be the most important feature. In particular, if the average number of fitness evaluations required to solve the problems in a set is far beyond the computational capabilities of one's computer, then one may become more interested in other features of the distribution of performance such as its variance or its tails. For example, consider two algorithms with the same mean number of fitness evaluations. Let us assume that the first algorithm has a large variance and/or fat tails in its performance across a set of problems and the second one has a very small variance in its performance distribution. It is clear that, thanks to its lower tail, the first algorithm would be able to solve at least a subset of the problems in a set, even if the computer used had insufficient power to solve the average problem. On the contrary, the second algorithm could not solve any problem with the available resources. Most people would consider the first algorithm more useful than the second. This illustrates the fact that in difficult domains knowing that a set of problems is closed under permutation and NFL applies is unlikely to be enough to draw conclusions about whether in practice superior algorithms exist. So, understanding the characteristics of the distribution of performance of GP algorithms (e.g., via higher order moments) should be an objective of future research.

4 The next decade: new directions

We now turn our attention to the most recent directions for theoretical research in GP.

4.1 Convergence proofs and computational complexity

While often Markov transition matrices describing EAs (including GP) are very large and difficult to manage numerically, it is sometimes possible to establish certain properties of such matrices theoretically. An important property in relation to the ability of an algorithm to reach all possible solutions to a problem (given enough time) is the property of *ergodicity*. What it means in practice is that there is some finite integer k such that all elements of M^k are non-zero.

This means that regardless of where the EA starts, the probability of reaching a solution in k or more iterations is bigger than zero. As a result, given enough generations the algorithm is guaranteed to find a solution.

In the context of EAs acting on traditional fixed-length representations, this has lead to proving that EAs with non-zero mutation rates are global optimisers with guaranteed convergence [112]. These results have been extended to more general search spaces [113]. Markov chain models of some forms of GP have also been recently derived (cf. Sect. 2.2 and [10]). So, a reasonable future step for GP theory would be to attempt to formally investigate under what conditions a GP system is guaranteed to find solutions.

Naturally, the calculations involved in setting up and studying these models should be expected to be of considerable mathematical complexity. This is particularly true if one considers infinite search spaces (which is required to model the traditional form of GP where operators can create trees of potentially any size). However, the task is not impossible. Indeed in [114] Schmitt and Droste provided a proof of convergence for a theoretical GP system where crossover and mutation rates are progressively reduced to zero. While their GP system has never been implemented and is dissimilar to current forms of GP, [114] provides a series of leads on how one could prove convergence for other forms of GP.

Despite the mathematical obstacles of proving it, it is actually surprisingly easy to find good reasons in support of the conjecture that, with minor modifications, the traditional form of GP is guaranteed to visit a solution to a problem, given enough generations. In fact, if one uses mutation and the mutation operator is such that, albeit with a very low probability, any point in the search space can be generated by mutating any other point, then it is clear that sooner or later the algorithm will visit a solution to the problem. Thus, in order to ensure that GP has a guaranteed asymptotic ability to solve all problems, all we need is to add to the mix of operators already present in a GP system some form of mutation that has a non-zero probability of generating any tree in the search space. One such mutation operator is the sub-tree mutation operator.

While asymptotic convergence to solutions is important from a theoretical standpoint, how quickly the GP system moves towards solutions is very important for practitioners. For example, one might be more interested in knowing that a GP system has a certain probability of finding a solution in N generations rather than knowing that it has a 100% probability of success in infinite generations. Thus, for example, computing the expected first visit time for a solution or the second largest eigenvalues of the Markov transition matrix (which determines the speed at which convergence is approached) would be important. There are new results from random graph theory and rapidly mixing Markov chains which may give interesting approximations (e.g., see [115]).

An alternative approach is offered by computational complexity (the "big O" notation), which is typically used to evaluate the scalability of classical algorithms. In the past decade or so theoreticians have started to apply computational complexity techniques to a variety of EAs. Precise bounds on the expected run time for a variety of EAs have been obtained by using such techniques [116–123]. Typically, this has been done only for specific classes of functions although there are some exceptions where run-time bounds have been derived for more general combinatorial optimisation problems [124–131].

To the best of our knowledge, nobody has attempted to apply computational complexity techniques to search in program spaces. Given the trajectory of such techniques, however, it is clear that it is only a question of time before this happens. Obviously, initially we should not expect to see models of the run time of a typical GP system with a large population, selection, crossover and mutation: this might barely be possible within our ten-year horizon. Nevertheless, in the next few years we expect to see computational complexity techniques being used to model simpler GP systems, perhaps GP systems based on mutation and stochastic hill-climbing.

4.2 Taxonomies

In many sciences a large part of theory is associated with taxonomy—classification with respect to natural relationships. In EAs, various high-level taxonomic labels are at our disposal, such as GP, GAs, evolution strategies, etc. The differences between these classes of algorithms are related to representations, genetic operators, selection schemes, etc. While these may have been clearly distinct two or more decades ago, today the degree of overlap between these categories is remarkable. This suggests they are not ideal for taxonomic purposes. Yet, most books and survey papers on EAs follow this high-level taxonomy and then further divide each group of algorithms based on their static characteristics. That is, the next level of the taxonomy is based on what individuals represent, on whether or not the representation is of fixed-size, and so on.

Similarly, it is common to distinguish different classes of population-based program-induction algorithms, such as GP, linear GP, Cartesian GP, Push-GP, Gene Expression GP, etc., based on differences in representations, operators, genotype-phenotype map, etc. That is, *many current taxonomies are based on algorithmic similarities and differences rather than behaviour.*

Whether this way of classifying algorithms is optimal, or even useful other than in an historic sense is debatable. Taxonomies allow us to understand commonality between different things as well as their differences. The classic example is the periodic table from Chemistry. It was initially constructed to reflect empirical, phenomenological, chemical facts until atomic theory gave it a firmer foundation. Can we formulate a periodic table for GP (and other EAs)? Can it just be based on form or should it also consider behaviour? If nothing like this exists that would be deeply worrying. The absence of any classification would mean that the theoretical treatment of each and every EA and/or problem would have to be different. It is clear however that there is commonality. The question is more—can it be formalised?

For program induction, one might think that building a taxonomy would require the classification of different GP systems. A GP system alone, however, is in some sense a "black box" that takes a "problem" (usually a fitness function) as input and, after a certain amount of processing, gives some output (typically the best individual found during a run). That is, a GP system alone does not fully specify a dynamics: we also need to specify a problem to obtain a complete dynamical system. Given some initial conditions (e.g., an initial random population) the system can then move in its state space. Hence, a taxonomy based on behaviour must really partition the composite space of (GP-system, problem) pairs. This does not mean that a taxonomy of GP systems (or more generally EAs) cannot exist. It simply means that one *may* need to specify to which class of problems this applies. Similarly a taxonomy of problems *may* need specify to which class of algorithms it applies. Note that we are careful not to rule out the possibility that research will be able to identify classes of GP problems which are intrinsically harder than other classes of GP problems, or GP algorithms that are intrinsically superior to other GP algorithms: after all in other domains computer scientist have been able to do this (e.g., P vs. NP).

Let us term $\mathcal{E}$ the space of (GP-system, problem) pairs. In principle one could put a metric on $\mathcal{E}$ and talk about how close one algorithm/problem pair is to another. For example, we could think of two elements of $\mathcal{E}$ as being close if they lead to similar behaviour. Of course, to do this one must define similarity measures for behaviours. At any rate, continuity on $\mathcal{E}$ would lead one to believe that algorithms with similar parameter values should normally behave similarly in relation to problems and *vice versa*. Naturally we might also expect sudden changes at singularities and/or phase transitions. Enormous benefits would accrue from knowing domains of continuity and their boundaries in $\mathcal{E}$. Also, knowing how rugged or smooth $\mathcal{E}$ is would be very important as it would enable us to estimate the validity of various models or approximations.

Obviously, to answer the above we need to have a formalism within which we can work—schema theories and Markov chains are just two possibilities. We suspect that progress will require a fuller characterisation of the sampling behaviour of different operators and finding a way of checking whether this behaviour matches the shape of a particular fitness landscape and to which degree. In its turn, this might require finding common languages in which to express the characteristics of both problems and algorithms. Do such languages exist?

It may well be that an exact common language does not really exist and that approximate models of algorithms and problems will be required to express both in the same language. As we mentioned in Sect. 2.5, it is possible to use simple models of GP [86, 87] to make accurate predictions of performance and use them to construct useful taxonomies of evolutionary program-induction algorithms for specific classes of problems. A technique to produce taxonomies of problems in evolutionary computation (including GP) has also been proposed in [132, 133]. So, there seem to be reasonable hope to make substantial taxonomic progress in GP in the next decade.

4.3 Dynamic and/or self-adapting GP frameworks

Many characteristics of standard forms of GP are static, in the sense that they are fixed before a run starts and remain constant through it. These include, for instance, the population size, the language used to code individuals, the genetic operators, their rates of application and the fitness function used. So, theory has focused on modelling such static forms of GP. However, it is easy to imagine situations where a dynamic or even a self-adapting form of GP might be beneficial. For example, there is empirical evidence (e.g., [134, 135, 77]) that suggests that there are benefits when populations are allowed to shrink or increase in size dynamically. It is likely that in the future we will see more and more dynamic and self-adapting GP systems.

Theory can play an important role in understanding the behaviour and thus the benefits and drawbacks of these systems. It is certainly not impossible to analyse their properties. For example, as we have seen in Sect. 4.1, a proof of convergence can be given for a system where crossover and mutation rates are annealed to zero [114]. Also, schema equations and Markov chain models can trivially be adapted to the case of dynamically changing rates of application of genetic operators and selection pressures. It might be more difficult to adapt these approaches to systems where more radical changes to the algorithm occur during a run. Also systems that change according to some fixed schedule are likely to be easier to model than self-adaptive systems. Other areas of current GP theory would need to be revised to cope with these systems.

Theory can also have a different role. It can help us design more competent dynamic and/or adaptive GP system. For example, recently difficulty measures have been used to dynamically choose the population size for GP so as to make best use of resources [136]. Also, as shown in [137, 138], it is possible to use theory to design new algorithms where the number of individuals that are actually created and evaluated in a population is progressively shrunk within a run so as to make best use of fitness evaluations.

4.4 GP for dynamic problems

Today GP is mainly used to solve static problems, in the sense that the problem being solved does not change during the course of a run. However, dynamic environments are very common in real-life applications. All forms of artificial open ended evolution as well as on-line and life-long learning require the ability to rapidly identify new optimal solutions should the requirements of a problem change over time thereby rendering the current best solution(s) sub-optimal. While there has been considerable interest in dynamic problems within other forms of EA, there has been surprisingly little work on applying and explicitly studying GP in dynamic environments [139–147]. This seems to be an important gap, particularly considering that recent evolutionary biology simulations suggest that dynamic environments should help, and not hinder, evolution also in artificial systems like EAs [148, 149].

The main open issues in relation to the use of GP in dynamic environments are discussed in [85]. Here, we will focus only on one question: how can current GP theory be adapted to dynamic environments? Again, in principle, schema theories and Markov chains could easily be adapted to dynamic problems. However, obtaining solutions for (and more generally obtaining information about) the dynamics of a system would be considerably harder than it already is. It is clear that static environments are only a special case of dynamic ones. So, it is likely that GP systems will exhibit more complex and interesting behaviours in dynamic environments. This makes understanding them a very worthy goal for theoretical research.

4.5 Generalisation in GP

Only few papers dealing with generalisation in GP have appeared to date (see, for instance, [150–156]). Generally little attention has been devoted to the study of the theoretical aspects of generalisation, contrarily to what has happened for many other machine learning strategies, like, for instance, support vector machines (e.g., see [157]). Below we briefly review the only two theoretical frameworks we are aware of in this area.

A commonly held principle, among many researchers, is called *minimum description length* (see for instance [158]). This states that the best model for explaining a dataset is the one that minimises the amount of information needed to encode it. Zhang and Mühlenbein [59] have incorporated information theoretic and minimum description length ideas into GP fitness functions to provide a degree of regularisation and so avoid overfitting (and bloat, see Sect. 2.4).

Another theoretical contribution to generalisation in GP is represented by [159], which showed that, under certain conditions, the number of fitness cases needed in GP's fitness evaluation can be upper-bounded via statistical and information-theoretic considerations. Results indicated that the number of fitness cases needed for reliable function reconstruction in GP is well approximated by the entropy of the target function. Formal results were in good agreement with experimental evidence.

While this research is a very good start, theory on generalisation in GP has not developed nearly enough in the last decade. Thus, this remains an open issue in GP (see [85] for a more extensive discussion).

4.6 Modularity in GP

The construction of any highly complex system typically uses hierarchical, modular structures to manage and organise that complexity. Given the pervasiveness of hierarchical structure as an organisational tool in both biology and engineering, it is reasonable to expect that such modular structure could be valuable in GP as well. Consequently, this has been a subject of substantial empirical exploration from the early days of GP (e.g., see [2, 160–166]) and the source of many open issues, as discussed in [85]. For example, it was suggested in [3], and later confirmed in [167], that a certain degree of modularity might be a result of the action of crossover and other operators. However, there has been very little study of the subject of modularity from a theoretical point of view.

An important set of results that link modularity to expressiveness of a representation have been obtained by Woodward [168, 169]. He showed that in the absence of modularity different equally expressive primitive sets may implement different program functionalities with substantially different numbers of primitives. However, in the presence of modularity there is invariance with respect to changes of primitive sets in the complexity of functions (up to a constant). These results have also been extended to Cartesian GP [170] and also primitive recursive functions [171].

Many theoretical questions about modularity remain unanswered. Does modularity offer the potential to solve substantially more complex problems than we can currently solve? Would modularity provide better scalability for GP? For what classes of problems would modularity help or hinder the search? Can a theory be developed that would lead to a principled design of GP systems that naturally exhibit modularity, hierarchy and reuse? Can we define metrics that would tell us to what degree a system exhibits such properties and could, therefore, guide us in the design of new systems? Can we adapt preexisting theoretical approaches?

5 Conclusions

We have presented the current state of the art in GP theory as well as a discussion of possible directions for future theoretical research. We have reviewed a variety of approaches, such as the GP schema theory, the modelling of GP by means of Markov chains, the asymptotic convergence properties of GP, the search spaces and fitness landscapes associated with GP. We have also outlined attempts to understand what makes a problem hard or easy for GP, performance models of GP, bloat, and the no-free-lunch. All of these have had some success, particularly in the last few years. Progress is slow for theory and, so, we feel that in many of these areas we have only started scratching the surface. There are also many new research avenues that have just started to become clear, which are essentially unexplored and where substantial progress could be made in the next decade.

Many unanswered questions about GP, and more generally program induction, still remain. What goes wrong when GP cannot solve a problem? Is it possible to measure the suitability of a problem for GP via a rigorous, reliable and predictive measure of problem hardness? Do we have a complete understanding of the biases of the search operators? Is bloat now a solved problem? Can we define a clear and formal relationship between bloat and over-fitting? Can we formulate a periodic table for GP systems and problems? Is there a common languages in which to express the characteristics of both GP problems and GP systems? Can we derive theoretical results that will guide the design of GP systems? Which GP representation, operators, selection strategy, population size, number of generations, anti-bloat method, etc. should one use for a given problem or a given class of problems? How can current GP theory be adapted to dynamic environments? Does modularity offer better scalability for GP? How can GP be encouraged to develop modular and hierarchical solutions? GP was inspired by evolution in nature, can GP theory offer something back to its "older, bigger brother"—population biology? Can we make our theoretical frameworks clearer, simpler and more accessible to non-theoreticians? These are just a subset of the challenges that researchers in the GP field will need to face in the next decade.

Of course, our analysis is by no means exhaustive, though we believe there are enough directions for research in this paper to keep very many GP theoreticians busy for many years to come. Hopefully, it may help to stimulate a new generation of theoreticians as it is currently stimulating ourselves, our collaborators and our students. We strongly believe that GP benefits from an interdisciplinary approach and we would hope that more talented researchers from other fields will enter the fray bringing with them their own points of view and toolboxes.

Acknowledgments The authors would like to thank Julian Miller and the referees of this paper for their extremely useful feedback. Lee Spector is also gratefully thanked.

References

1. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, MA, 1992)
2. J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press, Cambridge MA, May 1994)
3. 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, CA, 1998)
4. W.B. Langdon, R. Poli, *Foundations of Genetic Programming* (Springer, Berlin, 2002)
5. R. Poli, W.B. Langdon, N.F. McPhee, A field guide to genetic programming (2008). Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (with contributions by J. R. Koza).
6. R. Poli, Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. Genet. Program. Evol. Mach. **2**, 23–163, (2001)
7. N.F. McPhee, R. Poli, J.E. Rowe, A schema theory analysis of mutation size biases in genetic programming with linear representations. in *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, (COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea), (IEEE Press, 27–30 May 2001) pp. 1078–1085
8. R. Poli, N.F. McPhee, General schema theory for genetic programming with subtree-swapping crossover: Part I. Evol. Comput. **11**, 53–66 (2003)
9. R. Poli, N.F. McPhee, General schema theory for genetic programming with subtree-swapping crossover: Part II. Evol. Comput. **11**, 169–206 (2003)

10. R. Poli, N.F. McPhee, J.E. Rowe, Exact schema theory and Markov chain models for genetic programming and variable-length genetic algorithms with homologous crossover. Genet. Program. Evol. Mach. **5**, 31–70 (2004)
11. 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*, ed. by W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith. vol. 2 (Morgan Kaufmann, Orlando, FL, 13–17 July 1999), pp. 1135–1142
12. J.F. Miller, P. Thomson, Cartesian genetic programming. in *Genetic Programming, Proceedings of EuroGP'2000*, ed. by R. Poli, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty. LNCS, vol. 1802, (Springer, Edinburgh, 15–16 April 2000), pp. 121–132
13. C. Ryan, J.J. Collins, M. O'Neill, Grammatical evolution: Evolving programs for an arbitrary language. in *Proceedings of the First European Workshop on Genetic Programming*, ed. by W. Banzhaf, R. Poli, M. Schoenauer, T.C. Fogarty. LNCS, vol. 1391, (Springer, Paris, 14–15 April 1998), pp. 83–95
14. M. O'Neill, C. Ryan, Grammatical evolution. IEEE Trans. Evol. Comput. **5**, 349–358 (2001)
15. L.J. Fogel, A.J. Owens, M.J. Walsh, *Artificial Intelligence Through Simulated Evolution* (Wiley, New York, 1966)
16. R. Poli, C.R. Stephens, The building block basis for genetic programming and variable-length genetic algorithms. Int. J. Comput. Intell. Res. **1**(2), 183–197 (2005)
17. R.L. Riolo, U.-M. O'Reilly, T. McConaghy (eds.), *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, (Ann Arbor, Springer, 14–16 May 2009)
18. J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence* (MIT Press, Cambridge, 1992). First Published by University of Michigan Press 1975
19. D. Whitley, A genetic algorithm tutorial. Stat. Comput. **4**, 65–85 (1994)
20. C.R. Stephens, H. Waelbroeck, Effective degrees of freedom in genetic algorithms and the block hypothesis. in *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, ed. by T.Bäck (Morgan Kaufmann, East Lansing, 1997), pp. 34–40
21. C.R. Stephens and H. Waelbroeck, Schemata evolution and building blocks. Evol. Comput. **7**(2), 109–124 (1999)
22. L. Altenberg, Emergent phenomena in genetic programming. in *Evolutionary Programming—Proceedings of the Third Annual Conference*, ed. by A.V. Sebald, L.J. Fogel, (World Scientific Publishing, San Diego, CA, 24–26 Feb 1994), pp. 233–241
23. U.-M. O'Reilly, F. Oppacher, The troubling aspects of a building block hypothesis for genetic programming. in *Foundations of Genetic Algorithms 3*, ed. by L.D. Whitley, M.D. Vose. (Morgan Kaufmann, Estes Park, CO, 31 July–2 Aug 1994), pp. 73–88, Published 1995
24. P.A. Whigham, A schema theorem for context-free grammars. in *1995 IEEE Conference on Evolutionary Computation*, vol. 1, (IEEE Press, Perth, Australia, 29 Nov–1 Dec 1995), pp. 178–181
25. R. Poli, W.B. Langdon, A new schema theory for genetic programming with one-point crossover and point mutation. in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, ed. by J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo. (Morgan Kaufmann, Stanford University, CA, 13–16 July 1997), pp. 278–285
26. J.P. Rosca, Analysis of complexity drift in genetic programming. in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, ed. by J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo. (Morgan Kaufmann, Stanford University, CA, 13–16 July 1997), pp. 286–294
27. R. Poli, Hyperschema theory for GP with one-point crossover, building blocks, and some new results in GA theory. in *Genetic Programming, Proceedings of EuroGP'2000*, ed. by R. Poli, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty. LNCS, vol. 1802, (Springer, Edinburgh, 15–16 April 2000), pp. 163–180
28. R. Poli, Exact schema theorem and effective fitness for GP with one-point crossover. in *Proceedings of the Genetic and Evolutionary Computation Conference*, ed. by D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, H.-G. Beyer. (Morgan Kaufmann, Las Vegas, July 2000), pp. 469–476
29. R. Poli, Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. Genet. Program. Evol. Mach. **2**(2), pp. 123–163 (2001)
30. T.E. Davis, J.C. Principe, A Markov chain framework for the simple genetic algorithm. Evol. Comput. **1**(3), pp. 269–288 (1993)

31. B. Mitavskiy, J. Rowe, Some results about the Markov chains associated to GPs and to general EAs. Theor. Comput. Sci. **361**, pp. 72–110 (2006)
32. W.B. Langdon, Convergence rates for the distribution of program outputs. in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, ed. by W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, N. Jonoska. (Morgan Kaufmann, New York, 9–13 July 2002), pp. 812–819
33. W.B. Langdon, How many good programs are there? How long are they? in *Foundations of Genetic Algorithms VII*, ed. by K.A. De Jong, R. Poli, J.E. Rowe (Morgan Kaufmann, Torremolinos, Spain, 4–6 Sept 2002), pp. 183–202, Published 2003
34. W.B. Langdon, The distribution of reversible functions is Normal. in *Genetic Programming Theory and Practise*, ed. by R.L. Riolo, B. Worzel (Kluwer, Dordrecht, 2003) Chap. 11, pp. 173–188
35. W.B. Langdon, Convergence of program fitness landscapes. in *Genetic and Evolutionary Computation—GECCO-2003*, ed. by E. Cantú-Paz, J.A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M.A. Potter, A.C. Schultz, K. Dowsland, N. Jonoska, J. Miller. LNCS, vol. 2724, (Springer, Chicago, 12–16 July 2003), pp. 1702–1714
36. W.B. Langdon, The distribution of amorphous computer outputs. in *The Grand Challenge in Non-Classical Computation: International Workshop*, ed. by S. Stepney, S. Emmott (York, UK 18–19 April 2005).
37. R. Poli and W.B. Langdon, Efficient Markov chain model of machine code program execution and halting. in *Genetic Programming Theory and Practice IV*, ed. by R.L. Riolo, T. Soule, B. Worzel (Ann Arbor: Springer, 11–13 May 2006), vol. 5 of *Genetic and Evolutionary Computation*, Chap. 13
38. J. Woodward, Evolving turing complete representations. in *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, ed. by R. Sarker, R. Reynolds, H. Abbass, K.C. Tan, B. McKay, D. Essam, T. Gedeon (IEEE Press, Canberra, 8–12 Dec 2003), pp. 830–837
39. 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*, ed. by M. O'Neill, L. Vanneschi, S. Gustafson, A.I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, E. Tarantino. Lecture Notes in Computer Science, vol. 4971(Springer, Naples, 26–28 Mar. 2008), pp. 134–145
40. J.R. Woodward, R. Bai, Canonical representation genetic programming. in *GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, ed. by L. Xu, E.D. Goodman, G. Chen, D. Whitley, Y. Ding. (ACM, Shanghai, China, 12–14 June 2009), pp. 585–592
41. W.A. Tackett, Recombination, Selection, and the Genetic Construction of Computer Programs. PhD thesis, University of Southern California, 1994
42. N.F. McPhee, J.D. Miller, Accurate replication in genetic programming. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, ed. by L. Eshelman. (Morgan Kaufmann, Pittsburgh, PA, 15–19 July 1995), pp. 303–309
43. P. Nordin, W. Banzhaf, Complexity compression and evolution. in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, ed. by L. Eshelman. (Morgan Kaufmann, Pittsburgh, PA, 15–19 July 1995), pp. 310–317
44. T. Blickle, L. Thiele, Genetic programming and redundancy. in *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, ed. by J. Hopf. (Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany), pp. 33–38, Max-Planck-Institut für Informatik (MPI-I-94-241), 1994
45. T. Soule, J.A. Foster, Removal bias: A new cause of code growth in tree based evolutionary programming. in *1998 IEEE International Conference on Evolutionary Computation*, (IEEE Press, Anchorage, AK, 5–9 May 1998), pp. 781–786
46. W.B. Langdon, R. Poli, Fitness causes bloat. in *Soft Computing in Engineering Design and Manufacturing*, ed. by P.K. Chawdhry, R. Roy, R.K. Pant (Springer, London, 23–27 June 1997), pp. 13–22
47. W.B. Langdon, T. Soule, R. Poli, J.A. Foster, The evolution of size and shape. in *Advances in Genetic Programming 3*, ed. by L. Spector, W.B. Langdon, U.-M. O'Reilly, P.J. Angeline (MIT Press, Cambridge, MA, June 1999), Chap. 8, pp. 163–190
48. W. Banzhaf, W.B. Langdon, Some considerations on the reason for bloat. Genet. Program. Evol. Mach. **3**, pp. 81–91 (2002)

49. J. Rosca, A probabilistic model of size drift. in *Genetic Programming Theory and Practice*, ed. by R.L. Riolo, B. Worzel (Kluwer, Dordrecht, 2003), Chap. 8, pp. 119–136
50. R. Poli, General schema theory for genetic programming with subtree-swapping crossover. in *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, vol. 2038, (Springer, Como, 18–20 April 2001)
51. R. Poli, N.F. McPhee, Covariant parsimony pressure in genetic programming. Tech. Rep. CES-480, Department of Computing and Electronic Systems, University of Essex, Jan 2008
52. G.R. Price, Selection and covariance. Nature **227**(5257), pp. 520–521 (1970)
53. W.B. Langdon, *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!, vol. 1 of Genetic Programming* (Kluwer, Boston, 24 April 1998)
54. R. Poli, W.B. Langdon, S. Dignum, On the limiting distribution of program sizes in tree-based genetic programming. in *Proceedings of the 10th European Conference on Genetic Programming*, ed. by M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A.I. Esparcia-Alcázar. Lecture Notes in Computer Science, vol. 4445 (Springer, Valencia, Spain, 11– 13 April 2007), pp. 193–204
55. S. Dignum, R. Poli, Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. in *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary computation*, ed. by D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J.A. Clark, D. Cliff, C.B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J.F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K.O. Stanley, T. Stutzle, R.A. Watson, I. Wegener (ACM Press, London, 7–11 July 2007), vol. 2, pp. 1588–1595
56. S. Dignum, R. Poli, Sub-tree swapping crossover and arity histogram distributions. in *Genetic Programming*, ed. by A.I. Esparcia-Alcázar, A. Ekárt, S. a Silva, S. Dignum, A. Şima Uyar. Lecture Notes in Computer Science, vol. 6021 (Springer, 2010), pp. 38–49
57. T. Soule, J.A. Foster, Effects of code growth and parsimony pressure on populations in genetic programming. Evol. Comput. **6**, pp. 293–309 (1998)
58. W.B. Langdon, Size fair and homologous tree genetic programming crossovers. Genet. Program. Evol. Mach. **1**, pp. 95–119 (2000)
59. B.-T. Zhang, H. Mühlenbein, Balancing accuracy and parsimony in genetic programming. Evol. Comput. **3**(1), pp. 17–38 (1995)
60. B.-T. Zhang, H. Mühlenbein, Evolving optimal neural networks using genetic algorithms with Occam's razor. Complex Syst. **7**, pp. 199–220 (1993)
61. B.-T. Zhang, P. Ohm, H. Mühlenbein, Evolutionary induction of sparse neural trees. Evol. Comput. **5**(2), pp. 213–236 (1997)
62. T. Soule, Code Growth in Genetic Programming. PhD thesis, University of Idaho, 15 May 1998
63. R. Poli, N.F. McPhee, Parsimony pressure made easy. in *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, (ACM, Atlanta, GA, 2008), pp. 1267–1274
64. R. Poli, A simple but theoretically-motivated method to control bloat in genetic programming. in *Genetic Programming, Proceedings of the 6th European Conference, EuroGP 2003*, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa. LNCS, (Springer, Essex, UK, 14–16 April 2003), pp. 211–223
65. R. Poli, Covariant Tarpeian method for bloat control in genetic programming. in *Genetic Programming Theory and Practice VIII.* Genetic and Evolutionary Computation, ed. by R.L. Riolo et al. (Springer, Ann Arbor, 2010) (in press)
66. S. Dignum, R. Poli, Operator equalisation and bloat free GP. in *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, ed. by M. O'Neill, L. Vanneschi, S. Gustafson, A.I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, E. Tarantino. Lecture Notes in Computer Science, vol. 4971 (Springer, Naples, 26–28 March 2008), pp. 110–121
67. S. Silva, S. Dignum, Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP. in *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, ed. by L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, M. Ebner. LNCS, vol. 548 (Springer, Tuebingen, 15–17 April 2009), pp. 159–170
68. S. Silva, L. Vanneschi, Operator equalisation, bloat and overfitting: A study on human oral bioavailability prediction. in *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ed. by G. Raidl, F. Rothlauf, G. Squillero, R. Drechsler, T. Stuetzle, M. Birattari, C.B. Congdon, M. Middendorf, C. Blum, C. Cotta, P. Bosman, J. Grahl, J. Knowles, D. Corne, H.-G. Beyer, K. Stanley, J.F. Miller, J. van Hemert, T. Lenaerts, M. Ebner, J. Bacardit,

M. O'Neill, M. Di Penta, B. Doerr, T. Jansen, R. Poli, E. Alba (ACM, Montreal, 8–12 July 2009), pp. 1115–1122

69. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, Boston, 1989)
70. J. Horn, D.E. Goldberg, Genetic algorithm difficulty and the modality of the fitness landscapes. in *Foundations of Genetic Algorithms, 3*, ed. by D. Whitley, M. Vose (Morgan Kaufmann, Estes Park, CO, 1995), pp. 243–269
71. M. Mitchell, S. Forrest, J. Holland, The royal road for genetic algorithms: Fitness landscapes and GA performance. in *Toward a Practice of Autonomous Systems, Proceedings of the First European Conference on Artificial Life*, ed. by F.J. Varela, P. Bourgine (The MIT Press, Cambridge, MA, 1992), pp. 245–254
72. S. Forrest, M. Mitchell, What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation. Mach. Learn.**13**, pp. 285–319 (1993)
73. P.F. Stadler, Fitness landscapes. in *Biological Evolution and Statistical Physics*, ed. by M.Lässig, Valleriani. Lecture Notes Physics, vol. 585, (Springer, Heidelberg, 2002), pp. 187–207
74. K.E. Kinnear Jr., Fitness landscapes and difficulty in genetic programming. in *Proceedings of the First IEEE Conference on Evolutionary Computing* (IEEE Press, Piscataway, NY, 1994), pp. 142–147
75. N.I. Nikolaev, V. Slavov, Concepts of inductive genetic programming. in *Genetic Programming, Proceedings of EuroGP'1998*, ed. by W. Banzhaf et al. LNCS, vol. 1391 (Springer, 1998), pp. 49–59
76. T. Jones, Evolutionary Algorithms, Fitness Landscapes and Search. PhD thesis, University of New Mexico, Albuquerque, 1995.
77. L. Vanneschi, Theory and Practice for Efficient Genetic Programming. PhD thesis, Faculty of Sciences, University of Lausanne 2004
78. M. Tomassini, L. Vanneschi, P. Collard, M. Clergue, A study of fitness distance correlation as a difficulty measure in genetic programming. Evol. Comput. **13**, pp. 213–239 (2005)
79. L. Vanneschi, M. Clergue, P. Collard, M. Tomassini, S. Vérel, Fitness clouds and problem hardness in genetic programming. in *Genetic and Evolutionary Computation – GECCO-2004, Part II* ed. by K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P.L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, A. Tyrrell. Lecture Notes in Computer Science, vol. 3103 (Springer, Seattle, WA, 26–30 June 2004), pp. 690–701
80. L. Vanneschi, M. Tomassini, P. Collard, S. Vérel, Negative slope coefficient. A measure to characterize genetic programming. in *Proceedings of the 9th European Conference on Genetic Programming*, ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt. Lecture Notes in Computer Science, vol. 3905 (Springer, Budapest, Hungary, 10–12 April 2006), pp. 178–189
81. L. Vanneschi, Investigating problem hardness of real life applications. in *Genetic Programming Theory and Practice V*. Genetic and Evolutionary Computation. ed. by R.L. Riolo, T. Soule, B. Worzel (Springer, Ann Arbor, 17–19 May 2007), Chap. 7, pp. 107–125
82. S. Gustafson, L. Vanneschi, Operator-based distance for genetic programming: Subtree crossover distance. in *Genetic Programming, 8th European Conference, EuroGP2005*, ed. by M. Keijzer et al. Lecture Notes in Computer Science, LNCS 3447 (Springer, Lausanne, Switzerland, 2005), pp. 178–189
83. L. Vanneschi, S. Gustafson, G. Mauri, Using subtree crossover distance to investigate genetic programming dynamics. in *Genetic Programming, 9th European Conference, EuroGP2006*, ed. by P. Collet et al. Lecture Notes in Computer Science, LNCS 3905 (Springer, Budapest, Hungary, 2006), pp. 238–249.
84. S. Gustafson, L. Vanneschi, Operator-based tree distance in genetic programming. IEEE Trans. Evol. Comput. **12**, p. 4 (2008)
85. M. O'Neill, L. Vanneschi, S. Gustafson, W. Banzhaf, Open issues in genetic programming. Genet. Program. Evol. Mach. (2010) (this issue)
86. M. Graff, R. Poli, Practical model of genetic programming's performance on rational symbolic regression problems. in *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, ed. by M. O'Neill, L. Vanneschi, S. Gustafson, A.I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, E. Tarantino. Lecture Notes in Computer Science, vol. 4971 (Springer, Naples, 26–28 March 2008), pp. 122–133
87. M. Graff, R. Poli, Automatic creation of taxonomies of genetic programming systems. in *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, ed. by L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, M. Ebner. LNCS, vol. 5481 (Springer, Tuebingen, 15–17 April 2009), pp. 145–158

88. D. Wolpert, W. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, pp. 67–82 (1997)
89. D. Whitley, J.P. Watson, Complexity theory and the no free lunch theorem. in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, ed. by E.K. Burke, G. Kendall (Springer, US, 2005), Chap. 11, pp. 317–339
90. C. Schumacher, M.D. Vose, L.D. Whitley, The no free lunch and problem description length. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, ed. by L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, E. Burke (Morgan Kaufmann, San Francisco, CA, 7–11 July 2001), pp. 565–570
91. J.R. Woodward, J.R. Neil, No free lunch, program induction and combinatorial problems. in *Genetic Programming, Proceedings of EuroGP'2003*, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa. LNCS, vol. 2610 (Springer, Essex, 14–16 April 2003), pp. 475–484
92. W. Banzhaf, A. Leier, Evolution on neutral networks in genetic programming. in *Genetic Programming Theory and Practice III*, ed. by T. Yu, R. Riolo, B. Worzel (Ann Arbor: Springer, May 2005), vol. 9 of *Genetic Programming*, Chap.14, pp. 207–221
93. R. Poli, M. Graff, N.F. McPhee, Free lunches for function and program induction. in *FOGA '09: Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, ed. by I. Garibay, T. Jansen, R.P. Wiegand, A.S. Wu (ACM, Orlando, FL, 9–11 Jan 2009), pp. 183–194
94. W.A. Greene, A non-linear schema theorem for genetic algorithms. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, ed. by D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, H.-G. Beyer (Morgan Kaufmann, Las Vegas, NV, 10–12 July 2000), pp. 189–194
95. J. He, X. Yao, Towards an analytic framework for analysing the computation time of evolutionary algorithms. Artif. Intell. **145**(1–2), pp. 59–97 (2003)
96. W.B. Langdon, R. Poli, Why ants are hard. in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, ed. by J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, R. Riolo (Morgan Kaufmann, University of Wisconsin, Madison, WI, 22–25 July 1998), pp. 193–201
97. J.E. Rowe, N.F. McPhee, The effects of crossover and mutation operators on variable length linear structures. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, ed. by L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, E. Burke (Morgan Kaufmann, San Francisco, CA, 7–11 July 2001), pp. 535–542
98. N.F. McPhee, R. Poli, J. E. Rowe, A schema theory analysis of mutation size biases in genetic programming with linear representations. in *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, (Seoul, Korea), May 2001
99. R. Poli, N.F. McPhee, Exact GP schema theory for headless chicken crossover and subtree mutation. in *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, ed. by (IEEE Press, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27–30 May 2001), pp. 1062–1069
100. A. Moraglio, R. Poli, Geometric landscape of homologous crossover for syntactic trees. in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC-2005)*, ed. by (IEEE, Edinburgh, 2–4 Sept 2005) vol. 1, pp. 427–434
101. R. Poli, C.R. Stephens, A.H. Wright, J.E. Rowe, On the search biases of homologous crossover in linear genetic programming and variable-length genetic algorithms. in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, ed. by W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, N. Jonoska (Morgan Kaufmann, New York), pp. 868–876
102. R. Poli, J.E. Rowe, C.R. Stephens, A.H. Wright, Allele diffusion in linear genetic programming and variable-length genetic algorithms with subtree crossover. in *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi. LNCS, vol. 2278 (Springer, Kinsale, Ireland, 3–5 April 2002), pp. 212–227
103. S. Dignum, R. Poli, Sub-tree swapping crossover, allele diffusion and GP convergence. in *Parallel Problem Solving from Nature - PPSN X*, ed. by G. Rudolph, T. Jansen, S. Lucas, C. Poloni, N. Beume. LNCS, vol. 5199 (Springer, Dortmund, 13–17 Sept 2008), pp. 368–377

104. W.B. Langdon, W. Banzhaf, Repeated sequences in linear genetic programming genomes. Complex Syst. **15**(4), pp. 285–306 (2005)
105. W.B. Langdon, W. Banzhaf, Repeated patterns in tree genetic programming. in *Proceedings of the 8th European Conference on Genetic Programming*, ed. by M. Keijzer, A. Tettamanzi, P. Collet, J.I. van Hemert, M. Tomassini. Lecture Notes in Computer Science, vol. 3447 (Springer, Lausanne, Switzerland, 30 Mar–1 April 2005), pp. 190–202
106. A. Moraglio, Towards a Geometric Unification of Evolutionary Algorithms. PhD thesis, Computer Science and Electronic Engineering, University of Essex, 2007
107. J. Togelius, R. De Nardi, A. Moraglio, Geometric PSO + GP = particle swarm programming. in *Proceedings of the IEEE World Congress on Computational Intelligence* (IEEE Press, Hong Kong, 1–6 June 2008)
108. A. Moraglio, S. Silva, Geometric differential evolution on the space of genetic programs. in *Genetic Programming*, ed. by A.I. Esparcia-Alcázar, A. Ekárt, S. Silva, S. Dignum, A. Şima Uyar. Lecture Notes in Computer Science, vol. 6021 (Springer, 2010), pp. 171–183
109. N.F. McPhee, R. Poli, Using schema theory to explore interactions of multiple operators. in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, ed. by W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, N. Jonoska (New York, Morgan Kaufmann, 2002)
110. R. Poli, Recursive conditional schema theorem, convergence and population sizing in genetic algorithms. in *Foundations of Genetic Algorithms Workshop (FOGA 6)*, ed. by W.M. Spears W. Martin (Charlottesville, VA, 2000)
111. M. Graff, R. Poli, Practical models for the performance of evolutionary program induction algorithms and their extension to other learners and problem solvers. Artif. Intell. (2009) (submitted)
112. G. Rudolph, Convergence analysis of canonical genetic algorithm. IEEE Trans. Neural Netw. **5**(1), pp. 96–101 (1994)
113. G. Rudolph, Convergence of evolutionary algorithms in general search spaces. in *Int. Conf. Evol. Comput*, pp. 50–54 (1996)
114. L.M. Schmitt, S. Droste, Convergence to global optima for genetic programming systems with dynamically scaled operators. in *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ed. by M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M.V. Butz, C. Coello Coello, D. Dasgupta, S.G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, F. Rothlauf, C. Ryan, D. Thierens (ACM Press, Seattle, WA, 8–12 July 2006) vol. 1, pp. 879–886
115. A. Coja-Oghlan, C. Cooper, A.M. Frieze, An efficient sparse regularity concept. in *SODA*, ed. by C. Mathieu (SIAM) pp. 207–216, 2009
116. S. Droste, T. Jansen, I. Wegener, A rigorous complexity analysis of the (1 + 1) evolutionary algorithm for separable functions with boolean inputs. Evol. Comput. **6**(2), pp. 185–196 (1998)
117. S. Droste, T. Jansen, I. Wegener, On the analysis of the (1+1) evolutionary algorithm. Theor. Comput. Sci. **276**(1–2), pp. 51–81 (2002)
118. I. Wegener, On the expected runtime and the success probability of evolutionary algorithms. in *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2000)*, ed. by U. Brandes, D. Wagner. Lecture Notes in Computer Science, vol. 1928 (Springer, Konstanz, Germany, June 15–17, 2000), pp. 1–10
119. T. Jansen, K.A. De Jong, I. Wegener, On the choice of the offspring population size in evolutionary algorithms. Evol. Comput. **13**(4), pp. 413–440 (2005)
120. C. Witt, Runtime analysis of the (mu + 1) EA on simple pseudo-boolean functions. Evol. Comput. **14**(1), pp. 65–86 (2006)
121. T. Jansen, I. Wegener, The analysis of evolutionary algorithms—A proof that crossover really can help. Algorithmica **34**(1), pp. 47–66 (2002)
122. T. Storch, I. Wegener, Real royal road functions for constant population size. Theor. Comput. Sci. **320**(1), pp. 123–134 (2004)
123. T. Jansen, I. Wegener, Real royal road functions—where crossover provably is essential. Discrete Appl. Math. **149**(1–3), pp. 111–125 (2005)
124. G.H. Sasaki, B. Hajek, The time complexity of maximum matching by simulated annealing. J. ACM **35**(2), pp. 387–403 (1988)
125. O. Giel, I. Wegener, Evolutionary algorithms and the maximum matching problem. in *STACS*, pp. 415–426 (2003)

126. F. Neumann, I. Wegener, Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theor. Comput. Sci. **378**(1), pp. 32–40 (2007)
127. J. Scharnow, K. Tinnefeld, I. Wegener, The analysis of evolutionary algorithms on sorting and shortest paths problems. J. Math. Model. Algorithm. **3**, 349–366 (2004)
128. S. Baswana, S. Biswas, B. Doerr, T. Friedrich, P.P. Kurur, F. Neumann, *Computing Single Source Shortest Paths Using Single-Objective Fitness* (ACM, New York, NY, 2009), pp. 59–66
129. C. Horoba, *Analysis of a Simple Evolutionary Algorithm for the Multiobjective Shortest Path Problem* (ACM, New York, NY, 2009), pp. 113–120
130. T. Storch, *How Randomized Search Heuristics Find Maximum Cliques in Planar Graphs* (ACM, New York, NY, 2006), pp. 567–574
131. J. Reichel, M. Skutella, *Evolutionary Algorithms and Matroid Optimization Problems* (ACM, New York, NY, 2007), pp. 947–954.
132. D. Ashlock, K. Bryden, S. Corns, On taxonomy of evolutionary computation problems, in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, (IEEE Press, Portland, Oregon, 2004), pp. 1713–1719.
133. D.A. Ashlock, K.M. Bryden, S. Corns, J. Schonfeld, An updated taxonomy of evolutionary computation problems using graph-based evolutionary algorithms. in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, ed. by G.G. Yen, L. Wang, P. Bonissone, S.M. Lucas (IEEE Press, Vancouver, 2006) pp. 403–410
134. D. Rochat, M. Tomassini, L. Vanneschi, Dynamic size populations in distributed genetic programming. in *Proceedings of the 8th European Conference on Genetic Programming*, ed. by M. Keijzer, A. Tettamanzi, P. Collet, J.I. van Hemert, M. Tomassini. Lecture Notes in Computer Science, vol. 3447 (Springer, Lausanne, Switzerland, 30 Mar–1 April 2005), pp. 50–61
135. M. Tomassini, L. Vanneschi, J. Cuendet, F. Fernandez, A new technique for dynamic size populations in genetic programming. in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, (IEEE Press, Portland, Oregon, 20–23 June 2004), pp. 486–493
136. D.C. Wedge, D.B. Kell, Rapid prediction of optimum population size in genetic programming using a novel genotype - fitness correlation. in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ed. by M. Keijzer, G. Antoniol, C.B. Congdon, K. Deb, B. Doerr, N. Hansen, J.H. Holmes, G.S. Hornby, D. Howard, J. Kennedy, S. Kumar, F.G. Lobo, J.F. Miller, J. Moore, F. Neumann, M. Pelikan, J. Pollack, K. Sastry, K. Stanley, A. Stoica, E.-G. Talbi, I. Wegener (ACM, Atlanta, GA, 2008), pp. 1315–1322
137. R. Poli, W.B. Langdon, Running genetic programming backward. in *Genetic Programming Theory and Practice III*. vol. 9 of *Genetic Programming*, ed. by T. Yu, R.L. Riolo, B. Worzel (Springer, Ann Arbor, 12–14 May 2005). Chap. 9, pp. 125–140
138. R. Poli, W.B. Langdon, Backward-chaining genetic programming. in *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, ed. by H.-G. Beyer, U.-M. O'Reilly, D.V. Arnold, W. Banzhaf, C. Blum, E.W. Bonabeau, E. Cantu-Paz, D. Dasgupta, K. Deb, J.A. Foster, E.D. de Jong, H. Lipson, X. Llora, S. Mancoridis, M. Pelikan, G.R. Raidl, T. Soule, A.M. Tyrrell, J.-P. Watson, E. Zitzler (ACM Press, Washington DC, 25–29 June 2005) vol. 2, pp. 1777–1778
139. I. Dempsey, Grammatical Evolution in Dynamic Environments. PhD thesis, University College Dublin, 2007
140. N. Wagner, Z. Michalewicz, M. Khouja, R.R. McGregor, Time series forecasting for dynamic environments: The DyFor genetic program model. IEEE Trans. Evol. Comput. **11**, pp. 433–452 (2007)
141. J.V. Hansen, P.B. Lowry, R.D. Meservy, D.M. McDonald, Genetic programming for prevention of cyberterrorism through dynamic and evolving intrusion detection. Decision Support Syst. **43**, pp. 1362–1374 (2007). Special Issue Clusters
142. D. Jakobović, L. Budin, Dynamic scheduling with genetic programming. in *Proceedings of the 9th European Conference on Genetic Programming*, ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt. Lecture Notes in Computer Science, vol. 3905 (Springer, Budapest, Hungary, 10–12 April 2006), pp. 73–84
143. R.H. Kibria, Y. Li, Optimizing the initialization of dynamic decision heuristics in DPLL SAT solvers using genetic programming. in *Proceedings of the 9th European Conference on Genetic Programming*, ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt. Lecture Notes in Computer Science, vol. 3905 (Springer, Budapest, Hungary, 10–12 April 2006), pp. 331–340

144. W.B. Langdon, R. Poli, Genetic programming bloat with dynamic fitness. in *Proceedings of the First European Workshop on Genetic Programming*, ed. by W. Banzhaf, R. Poli, M. Schoenauer, T.C. Fogarty. LNCS, vol. 1391 (Springer, Paris, 14–15 April 1998), pp. 96–112
145. L. Vanneschi, G. Cuccu, A study of genetic programming variable population size for dynamic optimization problems. in *Proceedings of the 2009 International Conference on Evolutionary Computation (ICEC 2009), part of the International Joint Conference on Computational Intelligence (IJCCI 2009)*, ed. by A. Rosa et al. (2009)
146. L. Vanneschi, G. Cuccu, Variable size population for dynamic optimization with genetic programming. in *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ed. by G. Raidl, F. Rothlauf, G. Squillero, R. Drechsler, T. Stuetzle, M. Birattari, C.B. Congdon, M. Middendorf, C. Blum, C. Cotta, P. Bosman, J. Grahl, J. Knowles, D. Corne, H.-G. Beyer, K. Stanley, J.F. Miller, J. van Hemert, T. Lenaerts, M. Ebner, J. Bacardit, M. O'Neill, M. Di Penta, B. Doerr, T. Jansen, R. Poli, E. Alba (ACM, Montreal, 8–12 July 2009), pp. 1895–1896
147. T. Hu, W. Banzhaf, The role of population size in rate of evolution in genetic programming. in *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, ed. by L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, M. Ebner. LNCS, vol. 5481 (Springer, Tuebingen, 15–17 April 2009), pp. 85–96
148. W. Banzhaf, G. Beslon, S. Christensen, J. Foster, F. Képès, V. Lefort, J. Miller, M. Radman, J. Ramsden, Guidelines: From artificial evolution to computational evolution: A research agenda. Nat. Rev. Genet. **7**(9), pp. 729–735 (2006)
149. N. Kashtan, E. Noor, U. Alon, Varying environments can speed up evolution. Proc. Nat. Acad. Sci. **104**, pp. 13711–13716 (2007)
150. F.D. Francone, P. Nordin, W. Banzhaf, Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets. in *Genetic Programming: Proceedings of the first Annual Conference*, ed. by J.R. Koza et al. (MIT Press, Cambridge, 1996), pp. 72–80
151. W. Banzhaf, F.D. Francone, P. Nordin, The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. in *4th International Conference on Parallel Problem Solving from Nature (PPSN96)*, ed. by W. Ebeling et al. (Springer, Berlin, 1996), pp. 300–309
152. I. Kushchu, An evaluation of evolutionary generalization in genetic programming. Artif. Intell. Rev. **18**(1), pp. 3–14 (2002)
153. A.E. Eiben, M. Jelasity, A critical note on experimental research methodology in EC. in *Congress on Evolutionary Computation (CEC'02)*, (Honolulu, Hawaii, USA), (IEEE Press, Piscataway, NJ, 2002), pp. 582–587
154. L.E. Da Costa, J.-A. Landry, Relaxed genetic programming. in *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ed. by M. Keijzer et al. vol. 1, (ACM Press, Seattle, WA, 8–12 July 2006), pp. 937–938
155. C. Gagné, M. Schoenauer, M. Parizeau, M. Tomassini, Genetic programming, validation sets, and parsimony pressure. in *Genetic Programming, 9th European Conference, EuroGP2006*, ed. by P. Collet et al. Lecture Notes in Computer Science, LNCS 3905 (Springer, Berlin, Heidelberg, New York, 2006), pp. 109–120
156. L. Vanneschi, D. Rochat, M. Tomassini, Multi-optimization improves genetic programming generalization ability. in *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ed. by D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J.A. Clark, D. Cliff, C.B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J.F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K.O. Stanley, T. Stutzle, R.A. Watson, I. Wegener (ACM Press, London, 7–11 July 2007), vol. 2, pp. 1759–1759
157. A.J. Smola, B. Scholkopf, A Tutorial on Support Vector Regression. Tech. Rep. Technical Report Series - NC2-TR-1998-030, NeuroCOLT2, 1999
158. J. Rissanen, Modeling by shortest data description. Automatica **14**, pp. 465–471 (1978)
159. M. Giacobini, M. Tomassini, L. Vanneschi, Limiting the number fitness cases in genetic programming using statistics. in *Parallel Problem Solving from Nature - PPSN VII*, ed. by J.J. Merelo-Guervos, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, H.-P. Schwefel. Lecture Notes in Computer Science, LNCS no. 2439 (Springer, Granada, Spain, 7–11 Sept 2002), pp. 371–380
160. P.J. Angeline, J.B. Pollack, The evolutionary induction of subroutines. in *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, (Lawrence Erlbaum, Bloomington, IN, 1992), pp. 236–241

161. L. Spector, Evolving control structures with automatically defined macros. in *Working Notes for the AAAI Symposium on Genetic Programming*, ed. by E.V. Siegel, J.R. Koza (AAAI, MIT, Cambridge, MA, 10–12 Nov 1995), pp. 99–105
162. J.P. Rosca, D.H. Ballard, Discovery of subroutines in genetic programming. in *Advances in Genetic Programming 2*, ed. by P.J. Angeline, K.E. Kinnear, Jr. (MIT Press, Cambridge, MA, 1996) Chap. 9, pp. 177–202
163. G. Seront, External concepts reuse in genetic programming. in *Working Notes for the AAAI Symposium on Genetic Programming*, ed. by E.V. Siegel, J.R. Koza (AAAI, MIT, Cambridge, MA, 10–12 Nov 1995), pp. 94–98
164. I. Jonyer, A. Himes, Improving modularity in genetic programming using graph-based data mining. in *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, ed. by G.C.J. Sutcliffe, R.G. Goebel (American Association for Artificial Intelligence, Melbourne Beach, FL, 11–13 May 2006), pp. 556–561
165. E. Hemberg, C. Gilligan, M. O'Neill, A. Brabazon, A grammatical genetic programming approach to modularity in genetic algorithms. in *Proceedings of the 10th European Conference on Genetic Programming*, ed. by M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A.I. Esparcia-Alcázar. Lecture Notes in Computer Science, vol. 4445 (Springer, Valencia, Spain, 11–13 April 2007), pp. 1–11
166. N.F. McPhee, E.F. Crane, S.E. Lahr, R. Poli, Developmental plasticity in linear genetic programming. in *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ed. by G. Raidl, F. Rothlauf, G. Squillero, R. Drechsler, T. Stuetzle, M. Birattari, C.B. Congdon, M. Middendorf, C. Blum, C. Cotta, P. Bosman, J. Grahl, J. Knowles, D. Corne, H.-G. Beyer, K. Stanley, J.F. Miller, J. van Hemert, T. Lenaerts, M. Ebner, J. Bacardit, M. O'Neill, M. Di Penta, B. Doerr, T. Jansen, R. Poli, E. Alba (ACM, Montreal, 8–12 July 2009), pp. 1019–1026, Nominated for best paper award in the GP track
167. N. Kashtan, U. Alon, Spontaneous evolution of modularity and network motifs. Proc. Nat. Acad. Sci. USA **102**, pp. 13773–13778 (2005)
168. J.R. Woodward, Modularity in genetic programming. in *Genetic Programming, Proceedings of EuroGP'2003*, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa. LNCS, vol. 2610 (Springer, Essex, 14–16 April 2003), pp. 254–263
169. J.R. Woodward, Algorithm Induction, Modularity and Complexity. PhD thesis, School of Computer Science, The University of Birmingham, 2005
170. J.R. Woodward, Complexity and cartesian genetic programming. in *Proceedings of the 9th European Conference on Genetic Programming*,ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt. Lecture Notes in Computer Science, vol. 3905 (Springer, Budapest, Hungary, 10–12 April 2006), pp. 260–269
171. J.R. Woodward, Invariance of function complexity under primitive recursive functions. in *Proceedings of the 9th European Conference on Genetic Programming*, ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt. Lecture Notes in Computer Science, vol. 3905 (Springer, Budapest, Hungary, 10–12 April 2006), pp. 310–319