Genet Program Evolvable Mach (2010) 11:339–363
DOI 10.1007/s10710-010-9113-2

CONTRIBUTED ARTICLE

Open issues in genetic programming

Michael O'Neill · Leonardo Vanneschi · Steven Gustafson · Wolfgang Banzhaf

Received: 23 November 2009 / Revised: 21 April 2010 / Published online: 14 May 2010

Abstract It is approximately 50 years since the first computational experiments were conducted in what has become known today as the field of Genetic Programming (GP), twenty years since John Koza named and popularised the method, and ten years since the first issue appeared of the *Genetic Programming & Evolvable Machines* journal. In particular, during the past two decades there has been a significant range and volume of development in the theory and application of GP, and in recent years the field has become increasingly applied. There remain a number of significant open issues despite the successful application of GP to a number of challenging real-world problem domains and progress in the development of a theory explaining the behavior and dynamics of GP. These issues must be addressed for GP to realise its full potential and to become a trusted mainstream member of the computational problem solving toolkit. In this paper we outline some of the challenges and open issues that face researchers and practitioners of GP. We hope this overview will stimulate debate, focus the direction of future research to

M. O'Neill (✉)
Complex & Adaptive Systems Lab, School of Computer Science & Informatics, University College Dublin, Dublin, Ireland
e-mail: m.oneill@ucd.ie

L. Vanneschi
University of Milano-Bicocca, Milan, Italy
e-mail: vanneschi@disco.unimib.it

S. Gustafson
GE Global Research, Niskayuna, NY, USA
e-mail: steven.gustafson@research.ge.com

W. Banzhaf
Memorial University of Newfoundland, St. John's, NL, Canada
e-mail: banzhaf@mun.ca

deepen our understanding of GP, and further the development of more powerful problem solving algorithms.

Keywords Open issues · Genetic programming

1 Introduction

The term "Genetic Programming" (GP) has become shorthand for the generation of programs, code, algorithms and structures in general through the application of variation and selection, as motivated by biological evolution [22]. It is a subfield of the area of Automatic Programming, though usually more closely associated with Machine Learning (ML), with applications in classification, control and regression. The origins of GP go back at least to the 1950s (see e.g., Friedberg's 1958 and 1959 papers [35, 36]), and Turing and others had thought about the evolution of algorithms [31]. While the history of GP remains to be written, notable contributions have been made by Smith and Cramer in the 1980's [18, 120] before, in 1989, John Koza proposed the method that subsequently became known as standard GP [62, 64–67]. GP has since grown [9, 30, 75] and now encompasses a large collection of techniques to evolve programs (recently discussed in [104]).

GP is concerned with the behavior and evolution of software or algorithmic entities [6], and even the physical embodiment of these entities, such as the case may be in robots or electronic circuits. GP thus concerns itself with active entities that produce results, rather than passive ones being optimised. As a model induction method it is capable of uncovering both the structure and content of a model, performing an optimisation on both.

Since its early years, GP has been employed to solve many practical problems, both in industry and academia. In spite of its success in applications, and a considerable amount of theory that has been developed, GP has not yet reached the popularity of other ML methods. At the current time, GP does not seem to be universally recognized as a mainstream and trusted problem solving strategy, despite the fact that in the last decade GP has been shown to outperform some of these methods in many important applications (see for instance [4, 13, 28, 32, 56, 78, 80, 148, 150]). The resistance in the ML community to embrace GP might have to do with its roots in Darwinian thinking. To this day, there is widespread skepticism about evolutionary thought in many quarters, stalling ideas based on evolution from being adopted widely. Science in general as well as Engineering are not immune to these tendencies, and it can be safely assumed that part of the motivation behind the disregard for GP is caused by the unsettling consequences of accepting non-intentional mechanisms into the fold of ML. In the meantime, however, helping to bridge this gap GP has been integrated into MatLab [119] and is now part of Mathematica too [61], and a number of public-domain and commercial packages are available that offer GP, at least as part of the toolkit (e.g., [33, 37, 79, 83]).

There are very good reasons for our community to be optimistic. As our field matures, we are continuously making significant advances in a theory of GP. GP is solving challenging real-world problems, demonstrating routine, human-competitive performance, and in our universities GP is being integrated into undergraduate and postgraduate courses. By effective communication of these successes through the education of future generations of researchers and industry leaders, and publication in popular science and media outlets, the message of GP is spreading out into the wider community. We are now starting to benefit from this as anecdotal evidence, and indeed personal experience of some of the authors of this paper, suggests that GP is increasingly being adopted in a diverse array of commercial settings including design and engineering, telecommunications and networks, gaming and finance. The number and variety of recent patent applications which adopt GP also provides tangible evidence of its increased adoption and recognition [95].

Of course we still have a lot to learn about GP and how to improve this powerful method. The primary motivation for this article is to highlight and draw attention to several open issues that are in need of study and clarification. As well as identifying these open issues, we will discuss previous and related work, point out areas of potential and limitations, and suggest ideas for future research. We are deeply optimistic about the future of GP and we believe we are entering an exciting new phase in our field of study. This optimism is due to the increased visibility and adoption of GP in industry and the wider community and, for example, arising from continued advances in GP theory and the increasing ease of access to parallelism through multi-cores etc., which can be readily exploited by GP. We hope that this overview, which originates from a number of perspective panel discussions at the EuroGP series of conferences, which started in Paris in 1998 [10] will help researchers to focus on further improving the method and on broadening its appeal.

2 Open issues

In this section, we outline some of the significant open issues in the field of GP. The ordering of the issues presented does not reflect an order of importance. For the highlighted issues, we discuss related published material and we point out limitations, if any, and in some cases suggest ways to overcome them. While much valuable research has been and continues to be undertaken in the field, at present there is a rush to applications without a proper acknowledgement that we still have a lot to do to understand GP sufficiently to tackle real-world problems in a more principled manner.

2.1 Identifying appropriate representations for GP

Issue: *Identifying appropriate representations, ideally based on some measure of quality that captures the relationship between the fitness landscape and the search process.*

Difficulty: *Hard to impossible to identify an optimal representation, but given a better understanding of the relationship between representation and search, differentiation between alternatives may be possible.*

From the very earliest experiments in the automatic generation of executable structures [30] a variety of representations have been explored including binary string machine code [35], finite state automata [31], generative grammatical encodings [25, 97, 142, 143] and the dominant tree-based form popularised by Koza [62]. Numerous alternative representations have also been proposed, including graph [128], strongly-typed [86], linear [14], linear-tree [52], and linear-graph [53]. Not surprisingly given our knowledge of No Free Lunch [145] that no one search algorithm will be the best for all problems, different representations have demonstrated varying degrees of prowess depending on the problems to which they are applied. Generally comparisons between representations are limited to performance metrics such as the mean best fitness obtained and/or the number of runs that successfully find the ideal solution on a series of classic GP benchmark problems. As a field we are exposed to the danger of proposing infinite novel variations of representations without approaching this fundamental issue in a rigorous and principled manner. In general, we cannot currently answer questions like: What is the best representation for my current problem of interest? To answer such questions we need to undertake a formal analysis of representations that currently exist. In this manner we might determine their relative strengths and weaknesses, for example, in terms of their ability to facilitate navigation through the search space, their ability to automatically identify and manipulate modularity in a hierarchical manner, or in terms of their sufficiency to represent the necessary computational structures to tackle general programming problems.

One notable source of inspiration that the GP community can take is from the research that Rothlauf has undertaken into representations, albeit largely from a more traditional Genetic Algorithm (GA) perspective [113]. This analysis of representation has focused on issues such as locality, redundancy, and scaling, and Rothlauf and Oetzel have recently started applying these techniques to GP [114]. Given that fixed-length representations adopted by more traditional Evolutionary Algorithms such as GA's are a limited special case of their variable-length counterparts, the field of GP has the potential to generate substantial contributions to our understanding of representation in the broader field of Evolutionary Computation.

An an interesting angle that is perhaps unique to the GP method, is that it is capable of evolving structures and therefore aspects of its own representation. Examples of this include Langdon's research on evolving data structures [71] to Spector's investigations on "autoconstructive evolution" with Pushpop GP [124], which co-evolve the search operators in addition to the individuals. Exploring the space of alternative algorithms and representations is exemplified by recent research on GP Hyperheuristics. Just as there are infinite programs in an unbounded GP search space, there are an infinite number of potential algorithms to search this space of programs. To say that searching such a space is a significant challenge might be an understatement, however, it may be that the GP method itself holds the key to discovering appropriate representations hand in hand with the algorithms themselves. Hyperheuristics are demonstrating early potential to outperform classic GP, and in an exciting twist, theoretical analysis suggests that a Free Lunch may be possible through their adoption [101, 102].

A great deal of literature now exists in GP on the issue of syntax, and grammatical approaches to GP have highlighted some of the benefits that these "syntax-aware" forms of GP can gain. Indeed, another article in this Special Issue provides a review of this very topic [81]. Until quite recently the issue of semantics had been overlooked, and there is much promise in this early research to the development of a Semantic-Aware form of GP [11, 12, 39, 72, 82, 90, 91]. This approach tries to develop operators that respect the semantics of programs or solutions as well as providing better search with higher-level constructs that have more complex semantics like loops and recursion. Grammars themselves have also been used to encode semantic information in addition to the syntactic rules of the structure language [17, 57]. We predict that the most efficient forms of GP will combine awareness of both syntax and semantics.

2.2 Fitness landscapes and problem difficulty in GP

Issue: *Identifying how hard a particular problem, or problem instance, will be for some GP system, enabling a practitioner to make informed choices before and during application.*

Difficulty: *Hard. Problem difficulty relates not only to the issue of representation, but also to the choice of genetic operators and fitness functions.*

What do practitioners currently do to understand if GP (or better, a given GP configuration) is the right method to solve their problem? The common practice is to test some GP configurations by performing simulations. Depending on the complexity of the application, this testing process may be very time consuming and interpreting the results of this simulation phase can be difficult, given the non-deterministic nature of GP. Under this perspective, having well defined and reliable indicators of the ability of (a given) GP (configuration) to solve a problem would be outstandingly important.

Even though elements like the presence or absence of particular structures in the population undoubtedly affect the ability of GP of solving problems more than what happens for GAs or for other optimization heuristics [20, 21], since the early studies of problem difficulty in GP (that date back to the work of Kinnear [59] with the study of the fitness *auto-correlation function*) it looked clear that reliable indicators of problem hardness must be based on the concept of *fitness landscape* [126].

The fitness landscape metaphor [147] can be helpful to understand the difficulty of a problem, i.e. the ability of a searcher to find the optimal solution for that problem. Nevertheless, in general fitness landscapes cannot be drawn because of the huge dimension of the search space and because of the generally complicated structure of GP neighbourhoods. For this reason, it is important to define measures able to capture some important features of fitness landscapes that possibly have a direct relationship with the difficulty of the problem. Probably the two most interesting measures of problem hardness for GP, based on the concept of fitness landscape, that have been introduced so far are: *fitness-distance correlation* (FDC) [129, 130] and *negative slope coefficient* (NSC) [136].

FDC is based on the concept that what makes a problem easy or hard for a searching strategy is the relationship between fitness and distance to the goal. It is a reliable indicator of problem hardness for GP, but it has the obvious drawback of not being predictive, in the sense that the genotype of the global optimum must be known beforehand to be able to calculate it.

NSC is based on the idea that difficulty (or more precisely *evolvability*) depends on the relationship between the fitness of the various individuals and the one of their neighbours in the topological space induced the transformation operators used by the searching strategy (this relationship is often represented by so called *fitness clouds*). The NSC succeeds in correctly quantifying the difficulty of some GP benchmarks and also real-life problems, but has a number of known and well defined problems that prevent it from being widely diffused.

These two measures are also discussed in another article published in this special issue [109]. Besides the fact that both of them are based on samples of the search space (this problem is discussed in detail in [130]), one of the most serious problems of these two measures is that they do not model many of the characteristics that are typical of the GP process, and in particular they do not simulate the behaviour of the crossover operator. In order to fill this gap, a measure of similarity/dissimilarity between individuals bound to GP crossover has recently been defined [41, 42, 134]. Operator-bound distance measures typically work by counting the minimum number of applications of the operator needed to transform an individual into the other. While this is a reasonable task for GP mutation (the most common being some variations on the Levenshtein edit distance [40] and the structural distance [27]), this can be hard to obtain for GP crossover, because it would imply considering all the necessary next populations or, at least, to approximate them. For this reason, the similarity/dissimilarity measure defined in [41, 42, 134] works by calculating the probability of correctly applying the operator once. This measure was shown to have reasonable computational complexity, and thus to be usable in practice. Furthermore, this measure was successful in studying the trend of fitness distance correlation of populations during the evolution, in implementing fitness sharing and in quantifying population diversity. Finally, it has recently been successfully used to improve GP generalization ability by means of the introduction of population repulsors [133]. The definition of this measure opens an interesting number of research challenges for GP. For instance, it would be interesting to investigate adaptive controls for population size, operator application and selection pressure based on this notion of operator-based similarity.

The road to define reliable and really useful hardness indicators for GP is still long, and much research is still to be performed. Furthermore, a dynamic modification of the GP algorithm "on the fly", i.e., of the fitness function, the genetic operators or the representation. represents one of the most challenging open issues in GP. A first attempt in this direction has recently been presented in [140].

2.3 Static versus dynamic problems

Issue: *How does GP perform in dynamic problem environments? Should dynamic environments be adopted as standard?*

Difficulty: *Medium to hard. The diversity of possible dynamic environments that might exist, given the degree and frequency of changes that can occur (ranging from stochastic changes to the opposite end of the spectrum with fully deterministic changes), makes a general statement about performance or utility difficult. It may be possible to make stronger statements about certain categories of dynamic environments.*

Dynamic environments abound in many application domains where they offer particular challenges for all optimisation and problem solving methods. A well-known strategy for survival in dynamic environments is to adopt a population-based approach [15, 25, 89], where rather than maintaining a single candidate solution, a population of candidate solutions is employed. This allows a diversity of potential solutions to be maintained, which increases the likelihood that a sufficient solution exists at any point in time to ensure the survival of the population in the long term. Dynamic environments can exhibit changes in many different ways including the frequency and degree/size of change. The types of changes might range from relatively small smooth transitions to substantial perturbations in all aspects of the domain [15, 25, 89].

Given the power of the biological process of evolution to adapt to ever changing environments, it is surprising that the number of studies applying and explicitly studying their artificial counterpart of GP in dynamic environments have been small [25]. Despite the existence of a recent special issue in the Genetic Programming and Evolvable Machines Journal on Dynamic environments [149], none of the four articles actually dealt with GP directly. While some applications in dynamic environments have been undertaken (e.g., [43, 50, 139]), there has been little analysis of the behaviour of GP in these environments. Examples have examined bloat [76], constant generation [25], and variable size populations for dynamic optimization [48, 132]. Nevertheless, many of the theoretical tools used to formalize EAs, like schema theory or Markov chains, seem to naturally adapt to dynamic environments, and thus can be used in the future for more rigorous analysis on how and why GP works in those environments, as also suggested in the article [109].

A key issue to consider here is the suitability of a problem for GP. We must remember that the natural biological process of evolution results in populations of individuals that have a great capacity to adapt, and therefore, a robustness that enables their survival. From a problem solving perspective why should we expect evolution to be successful at producing *optimal* solutions to problems when the consequence of such a process is to produce entities that have a single objective, *to survive*? This means that the kinds of solutions produced are *good enough* to ensure survival relative to competitors and do not need to be the best possible solution at that point in time. When tackling real-world dynamic problems we must therefore shift our mindset from one of optimisation to one of survival, and it is problems with these dynamic characteristics that we should be applying GP to as these are the problems at which we might expect an evolutionary-inspired search process to truly excel above other methods.

Given this perspective, are we missing the boat by focusing on static problems where there is a single target? It may be that GP (amongst other Evolutionary

Algorithms) is especially suited to dynamic environments. Recent experiments in evolutionary biology simulations, what has been termed computational evolution [7], suggest that EC/evolution could work efficiently "because" of dynamic environments and not "despite of" them [55], and that modularity can spontaneously arise, promoting hierarchical decomposition of a problem, because the environment is dynamically changing over time [47]. Explicit studies with GP are required to determine if GP can actually reap benefits from these domains.

2.4 The influence of biology on GP

Issue: *How much detail is necessary to adopt from the biological paradigm of natural evolution? How much simplification and abstraction of processes is necessary?*

Difficulty: *Medium to hard. An analysis of the impact of simplification or complexification of the algorithms on improvement of behaviour is necessary. It might be difficult to clearly discern causes and effects.*

Do we use a "sufficient" set of features from biological evolution to embody its full potential in our artificial evolutionary process? The question can be answered with a resounding "no". While it is true that we shall always only be approaching biological evolution with computational systems, the early and crude forms of evolutionary processes gleaned from nature and put to use in Genetic and Evolutionary Computation are in no way sufficient to harvest the power demonstrated by natural evolution.

We thus need to go back frequently to the natural example of biological evolution and study what else can be learned, similar to biologists who for the most part are still learning about the complexities and intricacies of the evolutionary process in nature. We are not forced to copy or mimic as much as possible, but we should also not refrain from taking up more ideas from biology if they are useful, either.

In a 2006 paper one of the authors has argued together with others that there might be an entire new field of research developing at the intersection between biological evolution and computing which was termed "Computational Evolution" [7]. In that paper a whole spectrum of biological phenomena has been exposed which deserves the attention of computer scientists.

A couple of topics are worth discussion in the particular context of GP. One of the main issues of GP is scalability, i.e. the ability to provide algorithmic solutions to problems that are of substantial size/dimensionality, maybe requiring code of thousands of lines (or equivalent, in other representations).[1] We know that this is difficult today, as the GP process always tends to compromise between the goals set by a fitness function and the complexity of the program. This is indeed one of the strengths of GP, as it provides a natural engine for generalization. But in connection with a difficult task it produces obstacles, which can be overcome only with recipes for modularity and scalability in place.

[1] Note that this is not scalability in a more narrow sense discussed in Sect. 2.8. Rather it refers to a metalevel of characterization here.

Biological evolution teaches us that there was a key event in the history of life when multi-cellularity was invented. Multi-cellularity and the accompanying modularity of function at a high level are related to the “invention” of eukaryotic cell organization (with a cell kernel holding genetic information, and an outer area of cell function). It was with eukaryotic cell organization only that development, or the growth of multi-cellular organisms out of single-celled zygotes (fertilized eggs) was possible. In GP, development is high on the research agenda, and it is hoped that “generative systems” as they are called will provide a natural solution to the problem of modularity and scalability.

Development in Nature, however, does come with more complex hereditary processes, since multi-cellular organisms need a way to provide inheritance on the cellular level. A mechanism is needed that should be orthogonal to the process of genetic inheritance through DNA copying. This process is now called epigenetic inheritance, and is one of the main ingredients to developmental processes in nature. Epigenetic inheritance [49] works by switching on and off whole batteries of genes, which is useful during the growth phase of a multi-cellular organism, and can be used as well to provide intermediate adaptivity (with time-scales of 1–3 generations) for response to environmental trends. This additional way to confer information from generation to generation of individuals can be argued to be a side-effect of the invention of the eukaryotic genome and the subsequent reorganization of the cellular machinery.

So it can be safely predicted that epigenetic effects will be important if GP will adopt development as a scalability mechanism. Essentially that would entail that programs will need to have additional control mechanisms that will allow the turning on and off of particular functions, based on environmental and developmental signals.

The genotype-phenotype map has been the subject of continued interest in GP since approximately two decades. Often, developmental processes have been considered as part of this area. While we do not want to dispute that it makes sense to lump developmental processes together with more basic features of the genotype-phenotype map, development is not the only issue that needs to be addressed when studying genotype-phenotype maps. Closer to the issue is the problem of (sometimes called a contradiction between) robustness and evolvability. How can a system be robust to random changes (fluctuations in the environment, random impact of mutations, chance encounters) yet at the same time respond adaptively to trends and opportunities?

Biological systems face this challenge every day, and a consensus seems to emerge among biologists that the genotype-phenotype relation is a key ingredient in mediating this tension [60, 138]. Speaking in abstract terms, genotype-phenotype maps need the feature of being able to filter out random variations that happen on the genotypic level and only allow “productive” changes to reach the phenotype. It is as if it would need to dampen out higher frequencies/chaotic changes of the underlying genetic system and transform these changes into adaptations.

The general message of Biology to EC is that complexification of our algorithms is a key ingredient to closing the gap between Biology and EC. Recent investigations in this direction show considerable promise [88, 92]. This has the potential to provide answers to other questions, too, that come under the heading of open-ended evolution.

2.5 Open-ended evolution in GP

Issue: *Design an evolutionary system capable of continuously adapting and searching.*

Difficulty: *Medium to Hard. There is difficulty in defining clear criterion for measuring success, and whether or not open-ended evolution is required by GP practitioners.*

Can we achieve open-ended evolution with GP, where, for example, a GP system forms the core part of a financial trading system required to continuously learn without being switched off? Would it be possible that that same system can adapt over time to a completely different problem?

Notions of computational evolution can again provide a bridge between biology and EC, as exemplified by recent work of Moore and co-workers [87]. Essential ingredients of open-ended evolution are (i) a dynamically changing fitness landscape, (ii) availability of co-evolutionary processes, (iii) search with continuously injected randomness.

The latter point is important. How can a GP system be open to randomness and at the same time stabilize its best adapted solutions? Tiered systems whose basic layer consist of random solutions that are being promoted to higher layers based on their performance seems to be a feasible way to achieve this goal. Systems based on fitness layers or on age layers have shown considerable success in this context [45, 46].

2.6 Generalization in GP

Issue: *Defining the nature of generalization in GP to allow the community to deal with generalization more often and more rigorously, as in other ML fields and statistical analysis.*

Difficulty: *Medium. The hardness of defining generalization in a reliable and rigorous way is common with other ML paradigms. The lack of generalization in a GP system is often due to lack of prior planning by the practitioner.*

How to ensure that we evolve solutions with good properties of generalization, i.e. that do not overfit training data? Generalization is one of the most important performance evaluation criteria for artificial learning systems [85]. A large amount of literature and of well established results exist concerning the issue of generalization for many non-evolutionary ML strategies. It is the case, for instance, of Artificial Neural Networks (see, among the many other references, [117]), or Support Vector Machines (see for instance [121]). On the other hand, this issue in GP has not received the attention it deserves and only few papers dealing with the problem of generalization have appeared [26, 68]. A survey of the main contributions on generalization in GP has been done some years ago by Kushchu in [68]. In [34] the authors use what they called the “Compiling GP System” to compare its generalization ability with that of other ML paradigms and show in [8] the positive effect of an extensive use of the mutation operator on generalization in GP with sparse data sets. In [19], Da Costa and Landry have recently proposed a new GP model called Relaxed GP, showing its generalization ability. In [38], Gagné and coworkers have recently investigated two methods to improve generalization in GP-based learning: (1) the selection of the best-of-run individuals using a three data sets methodology, and (2) the application of parsimony pressure to reduce the complexity of the solutions.

A common agreement of many researchers is the so called minimum description length principle (see for instance [111]), which states that the best model is the one that minimizes the amount of information needed to encode it. In this perspective, preference for simpler solutions and overfitting avoidance seem to be closely related, given that it should be more likely that a complex solution incorporates specific information from the training set, thus overfitting it, compared to a simpler solution. But, as mentioned in [100], this argument should be taken with care as too much emphasis on minimizing complexity can prevent the discovery of more complex yet more accurate solutions. In fact, if considered at a superficial level, it may seem to suggest that overfitting is related to bloat in GP. Recent contributions clearly show, however, that GP systems can be defined that bloat and do not overfit and viceversa [118]. Thus, bloat and overfitting seem two independent phenomena.

Overfitting seems more related to the functional complexity of the proposed solutions [131, 137] rather than to their size. Of course, many different definitions of functional complexity can be considered, each of them having advantages as well as questionable aspects. This is the case, for instance, for concepts such as curvature, resilience or the degree of the approximating polynomial, like in [137]. It is possible that each one of these concepts has a different and interesting relationship to overfitting.

In this perspective, multi-optimization is becoming more and more popular to counteract overfitting in GP, where different expressions of the functional complexity of the solution or other criteria are used as further objectives [38, 135, 137]. This research, even though promising, is clearly preliminary and represents a noteworthy open issue in GP. Some theoretical aspects of generalization in GP and hints for future research along these lines are also discussed in the article [109].

The nature of generalization in GP also needs a very careful look. For instance, solving the 5-parity problem is not so general as solving for *n*-parity, with *n* being 1,2,3,... This truely requires generalization, and not just filling in (interpolation) for unseen data. This more ambitious type of generalization has not been tackled widely in the ML community, but it provides enormous potential for GP. We think that it is worth pointing out that these concepts apply to other artificial learning systems, too, and not just to GP.

2.7 GP benchmarks

Issue: *Is it possible to define a set of test problems that can be rigorously evaluated by the scientific community and then accepted as a more or less agreed upon set of benchmarks for experimental studies in GP?*

Difficulty: *Medium to high. Benchmark problems serve as a common language for a community when developing algorithms, and therefore a certain amount of inertia to not adopt better ones exists. Also, since GP is already a fairly complex algorithm, adopting overly simple benchmark problems enables overall lower complexity in the system, albeit at the expense of the significance of results.*

Other fields of Evolutionary Computation, like GAs or PSO, have a wide set of generally studied and agreed open benchmark functions. For instance, GAs' experimental comparisons often use a mix of synthetic benchmarks, standard test problems and provably difficult ones, like those presented in [1, 23, 84, 108]. Furthermore, in more recent literature several real-parameter function optimization problems have been proposed as standard benchmark functions, like Sphere, Schwefel, Rosenbrock, Rastrigin, etc. In [127] 25 benchmark functions have been selected, to provide a fair comparison between optimization algorithms. This benchmark suite contains, respectively, 5 unimodal and 20 multimodal functions, further divided into basic, expanded and hybrid composition functions. Twenty-two of these functions are non-separable, two are completely separable, and one is separable near the global optimum. This benchmark suite has been accepted as a more or less agreed-upon standard for testing real-parameter optimization algorithms.

In the early years, GP benchmarks have been limited to the set of problems presented by Koza in his book [63]: *k*-even parity problem, *h*-multiplexer, various forms of symbolic regression, the artificial ant on the Santa Fe trail and the intertwined spirals problem. More recently, the GP community has begun to use a larger set of test functions, for instance, the suite of UCI repository datasets (recently updated) [5]. Among them are trivial problems, like the IRIS dataset, but also more interesting ones, like the thyroid cancer datasets and many others. More recently GP experimental studies have appeared using classification of network intrusion (see for instance [99, 122]), etc.

Overall, we would classify currently used test functions in GP into three broad categories: regression, classification and design. For regression, several different kinds of functions have been used, e.g. sinus, polynomials, Mexican hat, Mackey-Glass time series, etc. For classification, one may quote the UCI examples, the intertwined spirals, the parity problems, the multiplexer, protein localization, etc. For design: Adder, multiplier, several other circuits, trusses, tables and other structures are the most used examples.

While it is true that these problems cover a set of different possible applications, this set of benchmarks is still restricted compared to other fields of Evolutionary computation, and, more importantly, lacks a rigorous evaluation. Some attempts of defining new and tunably difficult test problems for GP have been made in [110] with the introduction of Royal Trees and in [129, 130] where Goldberg's trap functions are extended to GP.

But still the goal of having a large set of benchmark functions of different nature, like the ones presented in [127] for real-values parameter optimization is not yet achieved. This is probably due to the larger complexity of GP compared to GAs or other methods for parameters optimization. For instance, the function set in current benchmarks is composed of functions of a similar nature which does not reflect the huge variety of possibilities offered by GP. A case in point is program evolution through patches recently becoming more prominent through works like [141]. This aspect of the potential of GP is not yet represented in benchmarks, as they are not typical for other machine learning approaches.

2.8 GP and modularity

Issue: *Define a clear measure of success for what it means to achieve Scalable GP, as well as Modularity.*

Difficulty: *Medium. Adopting practices from other computer science methods may be one source of tools. Other insights could be gleaned from Biology.*

How well does GP scale to problems of increasing complexity/difficulty? How can we improve scalability of GP? What is scalability for GP in the first place? Given that representations can evolve (see for example systems like PAM-DGP [144]), and the complexity of solutions can evolve as well, what is scalability in GP?

Individuals in classical GP are usually constructed from a primitive set which consists of a function set and a terminal set. An extension to this approach is the ability to define modules, which are in turn tree-based representations defined in terms of the primitives. This is a very important issue to improve GP expressiveness, code reusability and performance. The most well known of these methods is Koza's Automatically Defined Functions (ADFs) [63]. There have been a large number of studies focusing on modularity in GP, and we give a flavour of some of these here before going on to highlight important open issues which still remain despite this body of work.

The first step towards a theoretically motivated study of ADFs is probably represented by [112], where an algorithm for the automatic discovery of building blocks in GP called adaptive representation through learning is proposed. In the same year, Spector [123] introduced techniques to evolve collections of automatically defined macros and showed how they can be used to produce new control structures during the evolution of programs and Seront [116] extended the concept of code reuse in GP to the concept of generalization, showing how programs (or "concepts", using Seront's terminology) synthesised by GP to solve a problem can be reused to solve other ones. This aim is achieved by the creation of a concepts library. These concepts can then be injected in a new population in order to solve a problem that needs them.

Linear GP [14] has other ways to evolve modularity. By reusing contents of registers, memory in LGP can be considered a substitute for ADFs in tree-based GP. This functionality comes essentially for free, and provides a means to evolve more compact solutions that would be possible in classical GP. Further, in both of these representations, evolution clearly makes reuse a virtue. If one looks closely one can discover numerous pieces of code being repeatedly used in various places in the programs [70, 73].

For tree-based GP, Woodward [146] showed that for a given problem, the minimum number of nodes in the main tree plus the nodes in any modules is independent of the primitive set (up to an additive constant) and depends only on the function being expressed. This reduces the number of user defined parameters in the run and makes the inclusion of a hypothesis in the search space independent of the primitive set. Altenberg offers a critical analysis of modularity in evolution in [2], stating that the evolutionary advantages that have been attributed to modularity do not derive from modularity “per se”. Rather, they require that there be an “alignment” between the spaces of phenotypic variation, and the selection gradients that are available to the organism. Modularity in the genotype-phenotype map may make such an alignment more readily attained, but it is not sufficient; the appropriate phenotype-fitness map in conjunction with the genotype-phenotype map is also necessary for evolvability. This contribution is interesting and stimulating, but its applicability to GP remains an open question. In [51] the concept of ADFs is extended by using graph-based data mining to identify common aspects of highly fit individuals and modularising them by creating functions out of the subprograms identified. In [44] the authors state that the ability of GP to scale to problems of increasing difficulty operates on the premise that it is possible to capture regularities that exist in a problem environment by decomposition of the problem into a hierarchy of modules. Thus, they present a comparison of two modular Genetic Algorithms, one of which is a Grammatical GP algorithm, the meta-Grammar Genetic Algorithm (mGGA), which generates binary string sentences instead of traditional GP trees. The presented results demonstrate some limitations of the modular GA (MGA) representation and how the mGGA can overcome these.

Finally, what about emergent phenomena like “neutral code” [93] and repeated code [74]? Do they provide natural ways of organising modularity and re-use in GP? In a recent paper by Kashtan et al [54] naturally emerging modules have been demonstrated in the context of a GA in coherently changing environments. This particular feature of the environment was key to the ability of the algorithm to develop modules. So while the former emergent phenomena can be said to be the result of genetic operators (like crossover), the latter is a reflection of the environment. Yet both are important aspects of any GP system, and should deserve further investigation.

As can be seen there is a large and varied literature related to modularity in GP, however, our work is by no means finished, in fact we argue that the most interesting aspects of this work have yet to be explored. Although the use of modularity in GP has helped solving some problems that straight GP couldn’t in a fixed number of runs (or solved them more efficiently) and helped provide new data/control abstractions, for instance in the form of ADFs, some issues remain open.

Are ADFs necessary/sufficient as a formalism to help solve grand-challenge problems i.e. to provide scalability? And even more ambitiously: Can we achieve modularity, hierarchy and reuse in a more controlled and principled manner? Can Software Engineering provide insights and metrics on how to achieve this?

In order to be able to get a deeper insight of the real usefulness of modularity and code-reuse in GP, more theoretical studies are needed. For instance, given the strong relationship between the concepts of ADFs and building blocks (already pointed out by Rosca in [112]), one may try to develop a framework based on schema theory [77] in order to formalize the effects of the use of ADFs in GP.

2.9 The complexity of GP

Issue: *What is the best way to setup GP to tackle a problem? Can we define a universal measure of complexity for GP? How does one handle the structural complexity of GP?*

Difficulty: *Medium. Factoring in the special cases of evolving code, which may be compiled and interpreted could be challenging, as well as the considerations in evolutionary search.*

Like all Evolutionary Algorithms, GP is a complex system. Rather than this subsection being about a single open issue there are multiple issues that arise due to this complexity. In this section we touch on some of these issues.

For example, understanding the interplay of the suite of available search operators and their rates, coupled with initialisation, selection and replacement strategies, which might be employed in any one application is non-trivial. This complexity increases when one introduces the use of a genotype-phenotype map, which may or may not be itself modulated by a feedback loop to the environmental state. The ability to predict the behaviour of such a system, and thus the choice of the *optimal set* of algorithm components is next to impossible with current analytic techniques. These questions have given rise to the search for better GP algorithms using GP, i.e., meta-GP (e.g., [16, 94, 103]). A relatively new departure in the field, meta-GP has exhibited potential to find better GP algorithms to solve a class of problems (e.g. Travelling Salesman [58]), and some recent theoretical analysis suggests that the normal rules of the No Free Lunch theory may not apply to meta-search [101].

All of the above overlooks one of the most challenging aspects of the GP representation, its *structural complexity*. Unlike fixed-length Genetic Algorithms, out of necessity GP adopts variable-length individuals as the structure of the solution must be uncovered in parallel with the combinatorial search of the symbols which make up any one structure. How best to search this variable-length structure space is an open issue within GP.

Given the structural complexity of the GP representation, it follows that fixed-length structures adopted by most of the other forms of Evolutionary Computation are a special case of their variable-length cousins. GP might be considered a superset of EC in terms of representation. As such, there is the possibility that theoretical advances made in the field of GP will apply more widely to EC, and through research in GP theory may provide a *unified theory of EC*.

2.10 Miscellaneous issues

In the above we have highlighted and discussed some of the more significant open issues that in our opinions currently exist in the field of GP. The order of their presentation suggests no ranking of their relative importance, and the list is not meant to be exhaustive. To discuss all open issues facing our community would require a book, so to give as much breadth to our coverage as possible we present a list of additional miscellaneous open issues that were considered, and which also must be addressed by our community.

– *The Halting Problem*: Part of the standard toolkit in GP is the use of program constructs such as iterations, loops, functions (including the use of recursion), and memory. Their use can be problematic, especially when all aspects of their use are open to the mechanism of evolutionary search, as they can give rise to the creation of programs which do not stop executing. How to handle the halting problem is a practical problem faced by users of GP, and a number of strategies have been employed to prevent or at least detect non-halting programs in some limit. This is still an area open to investigation.
– *Domain Knowledge*: How much domain knowledge should we inject into our GP algorithms? What is an appropriate AI ratio? Koza introduced the notion of an AI ratio [67] to highlight the fact that for a minimum amount of domain knowledge supplied by the user (the intelligence) a high-return (the artificial) can be achieved by GP. He also notes that one of the aims of the fields of artificial intelligence and machine learning are to achieve human-competitive results automatically, and that this can be partly measured by the existence of a high AI ratio. In terms of how much domain knowledge to incorporate the answer, at least in part, is that it depends on the problem. For some problems it can be useful to throw away the rulebook and let evolution find alternative solutions. For example, in conceptual architectural design [96, 98] the architect may want to exploit the stochastic nature of evolutionary search so as to explore an unbiased diversity of form, whose construction is not dictated to by traditional rules or even the preconceptions of the architect. On the other hand, another architect may want to explore form which is grounded in these rules and domain knowledge. In the worst cases, by incorporating domain knowledge unfavourable bias may be introduced making search for solutions difficult or impacting solution generalisation.
– *GP usefulness for Biology*: Can GP inform behaviour of Biological Evolution? While we don't attempt to perfectly model biology, the adoption of increasingly realistic processes may cast light upon the inner workings of biological systems.
– *Constants in GP*: Koza might be quoted about constants in GP *"...the finding of numerical constants is a skeleton in the GP closet...[and an] area of research that requires more investigation..."* [29]. Ephemeral Random Constants (ERCs) are the standard approach to constants in GP, and a number of variations on the ERC approach have been proposed to overcome their limitations (e.g., [3, 115, 125]), however this is still an open area of investigation (e.g., [24]).

- *GP Theory*: The development of an exact and general schema theory for GP [77, 105–107] undoubtably represents a very significant result. A difficulty for some researchers and practitioners is how best to use findings such as these. What, if anything else, does a theory of GP need to tell us precisely to bridge this divide? What is the potential for GP theory to inform a general theory of EC?
- *Distributed models of GP*: Open questions in this area that deserve attention include how to decide upon an appropriate rate of migration (with respect to the number of individuals allowed to migrate and how often a migration event occurs) in order to maintain a balance between the diversity afforded by separate islands and an appropriate level of information transfer between the populations? What is an appropriate topology by which connections (migration pathways) between populations are structured? What's the best way to exploit multi-core and GPU hardware (e.g., see [69])?
- *Usability of GP*: Practitioners need an easily digestible form of GP. There are so many parameters, function and terminal set membership, fitness function authoring and design, choice of representations and operators, general evolutionary parameter settings. Design and development of easier to use and more intuitive software implementations deserve attention, such as "one button" GP [33].

3 Conclusions

In this article we stress a number of open issues in the field of Genetic Programming in order to provide a spotlight for discussion. We hope that this document stimulates a conversation on what issues are most significant and what if any are missing, and as such help to steer future research in our community to provide a deeper understanding, and thereby strengthen the Genetic Programming method.

Acknowledgments The impetus for this article arose out of the EuroGP 2008 debate on *Grand Challenges of Genetic Programming* which took place on 27 March 2008 at the Evo* event in Naples, Italy. In particular we thank the two other panel members, Nic McPhee and Riccardo Poli, and also the many members of the audience who participated in the debate. Many of these issues have been raised on multiple occasions at previous (and subsequent) EuroGP debates so this inspired us to put these ideas on paper to open the debate to a wider audience. MO'N acknowledges support of Science Foundation Ireland under Grant No. 08/IN.1/I1868. WB acknowledges support from the Canadian National Science and Engineering Research Council (NSERC) under discovery grant RGPIN 283304-07.

References

1. L. Altenberg, NK fitness landscapes. In *Section B2.7.2 in Handbook of Evolutionary Computation*, ed. by T. Back et al. (IOP Publishing Ltd and Oxford University Press, Bristol and Oxford, 1997), pp. B2.7:5–B2.7:10
2. L. Altenberg, Modularity in evolution: Some low-level questions. In *Modularity: Understanding the Development and Evolution of Complex Natural Systems*, ed. by D. Rasskin-Gutman, W. Callebaut (MIT Press, Cambridge, MA, 2004, in press)

3. P.J. Angeline, Two self-adaptive crossover operators for genetic programming. In *Advances in Genetic Programming 2, ch. 5*, ed. by P.J. Angeline, K.E. Kinnear, Jr. (MIT Press, Cambridge, MA, 1996), pp. 89–110
4. F. Archetti, S. Lanzeni, E. Messina, L. Vanneschi, Genetic programming for computational pharmacokinetics in drug discovery and development. Gene. Program. Evolvable Mach. **8**(4), 413–432 (2007, Dec). Special issue on medical applications of Genetic and Evolutionary Computation
5. A. Asuncion, D. Newman, *UCI Machine Learning Repository* (2007)
6. W. Banzhaf, Editorial introduction to the first issue. Genet. Program. Evolvable Mach. **1**, 5–6 (2000)
7. W. Banzhaf, G. Beslon, S. Christensen, J. Foster, F. Képès, V. Lefort, J. Miller, M. Radman, J. Ramsden, From artificial evolution to computational evolution: a research agenda. Nat. Rev. Genet. **7**(9), 729–735 (2006)
8. 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
9. 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)
10. W. Banzhaf, R. Poli, M. Schoenauer, T. Fogarty (eds.), *Proceedings of Genetic Programming, 1st European Workshop, EuroGP'98, Paris, France, April 14–15, 1998*, vol. 1391 of LNCS (Springer, Berlin, 1998)
11. L. Beadle, C. Johnson, Semantically driven crossover in genetic programming. In *Proceedings of the IEEE World Congress on Computational Intelligence (Hong Kong, 1–6 June 2008)*, ed. by J. Wang, (IEEE Computational Intelligence Society, IEEE Press, 2008), pp. 111–116
12. S. Bhattacharyya, O. Pictet, G. Zumbach, Representational semantics for genetic programming based learning in high-frequency financial data. In *Genetic Programming 1998: Proceedings of the 3rd Annual Conference (University of Wisconsin, Madison, WI, USA, 22–25 July 1998)*, 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, 1998), pp. 11–16
13. S. Bianco, F. Gasparini, R. Schettini, L. Vanneschi, An evolutionary framework for colorimetric characterization of scanners. In *International Workshop on Evolutionary Computation in Image Analysis and Signal Processing, EvoIASP 2008. Proceedings of Applications of Evolutionary Computing, EvoWorkshops 2008*, vol. 4974/2008 of Lecture Notes in Computer Science, LNCS, ed. by M. Giacobini et al. (Springer, Berlin, Heidelberg, New York, 2008), pp. 245–254
14. M. Brameier, W. Banzhaf, *Linear Genetic Programming*. No. XVI in Genetic and Evolutionary Computation (Springer, Berlin, 2007)
15. J. Branke, *Evolutionary Optimization in Dynamic Environments* (Kluwer, Dordrecht, 2001)
16. E.K. Burke, M.R. Hyde, G. Kendall, Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature—PPSN IX (Reykjavik, Iceland, 9–13 Sept 2006)*, vol. 4193 of LNCS, ed. by T.P. Runarsson, H.-G. Beyer, E. Burke, J.J. Merelo-Guervos, L.D. Whitley, X. Yao (Springer, 2006), pp. 860–869
17. R. Cleary, M. O'Neill, An attribute grammar decoder for the 01 multiconstrained knapsack problem. In *Evolutionary Computation in Combinatorial Optimization—EvoCOP 2005 (Lausanne, Switzerland, 30 March–1 April 2005)*, vol. 3448 of LNCS, ed. by G.R. Raidl, J. Gottlieb, (Springer, 2005), pp. 34–45
18. N.L. Cramer, A representation for the adaptive generation of simple sequential programs. In *Proceedings of the International Conference on Genetic Algorithms and Their Applications (Carnegie-Mellon University, Pittsburgh, PA, July 1985)*, ed. by J.J. Grefenstette, pp. 183–187
19. L.E. Da Costa, J.-A. Landry, Relaxed genetic programming. In *GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (Seattle, WA, USA, 8–12 July 2006)*, vol. 1, ed. byn M. Keijzer et al. (ACM Press, 2006), pp. 937–938
20. J.M. Daida, R. Bertram, S. Stanhope, J. Khoo, S. Chaudhary, O. Chaudhary, What makes a problem GP-hard? Analysis of a tunably difficult problem in genetic programming. Genet. Program. Evolvable Mach. **2**, 165–191 (2001)
21. J.M. Daida, H. Li, R. Tang, A.M. Hilss, What makes a problem GP-hard? Validating a hypothesis of structural causes. In *Genetic and Evolutionary Computation—GECCO-2003*, vol. 2724 of LNCS, ed. by E.C.-P. et. al. (Springer, Berlin, 2003), pp. 1665–1677

22. C. Darwin, *On the Origins of the Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life* (1859)
23. K. Deb, J. Horn, D. Goldberg, Multimodal deceptive functions. Complex Syst. **7**, 131–153 (1993)
24. I. Dempsey, M. O'Neill, A. Brabazon, Constant creation with grammatical evolution. Int. J. Innov. Comput. Appl. **1**(1), 23–38 (2007)
25. I. Dempsey, M. O'Neill, A. Brabazon, *Foundations in Grammatical Evolution for Dynamic Environments*, vol. 194 of Studies in Computational Intelligence (Springer, 2009, Apr)
26. A.E. Eiben, M. Jelasity, A critical note on experimental research methodology in EC. In *Congress on Evolutionary Computation (CEC'02) (Honolulu, HI, USA, 2002)* (IEEE Press, Piscataway, NJ, 2002), pp. 582–587
27. A. Ekárt, S.Z. Németh, Maintaining the diversity of genetic programs. In *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002 (Kinsale, Ireland, 3–5 Apr 2002)*, vol. 2278 of LNCS, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi (Springer, 2002), pp. 162–171
28. S.E. Eklund, Time series forecasting using massively parallel genetic programming. In *Proceedings of Parallel and Distributed Processing International Symposium (22–26 Apr 2003)*, pp. 143–147
29. M. Evett, T. Fernandez, Numeric mutation improves the discovery of numeric constants in genetic programming. In *Genetic Programming 1998: Proceedings of the 3rd Annual Conference (University of Wisconsin, Madison, WI, USA, 22–25 July 1998)*, 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, 1998), pp. 66–71
30. D. Fogel, Evolving computer programs. In *Evolutionary Computation: The Fossil Record*, ed. by D. Fogel (MIT Press, Cambridge, MA, 1998), ch. 5, pp. 143–144
31. L. Fogel, A. Owens, M. Walsh, *Artificial Intelligence through Simulated Evolution* (Wiley, New York, 1966)
32. C. Fonlupt, Solving the ocean color problem using a genetic programming approach. Appl. Soft Comput. **1**(1), 63–72 (2001, June)
33. F. Francone, *The discipulus owner's manual*. URL: http://www.rmltech.com/technology_overview.htm (2004)
34. 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 1st Annual Conference*, ed. by J.R. Koza et al. (MIT Press, Cambridge, 1996), pp. 72–80
35. R. Friedberg, A learning machine: Part 1. IBM J Res. Dev. **2**(1), 2–13 (1958)
36. R. Friedberg, B. Dunham, J. North, A learning machine: Part 2. IBM J. Res. Dev. 282–287 (1959)
37. C. Gagne, Open beagle. URL: http://www.beagle.gel.ulaval.ca (11 2007)
38. C. Gagné, M. Schoenauer, M. Parizeau, Tomassini M., Genetic programming, validation sets, and parsimony pressure. In *Genetic Programming, 9th European Conference, EuroGP2006*, Lecture Notes in Computer Science, LNCS 3905, ed. by P. Collet et al. (Springer, Berlin, Heidelberg, New York, 2006), pp. 109–120
39. D.E. Goldberg, U.-M. O'Reilly, Where does the good stuff go, and why? how contextual semantics influence program structure in simple genetic programming. In *Proceedings of the 1st European Workshop on Genetic Programming (Paris, 14–15 Apr 1998)*, vol. 1391 of LNCS, ed. by W. Banzhaf, R. Poli, M. Schoenauer, T.C. Fogarty, (Springer, 1998), pp. 16–36
40. S. Gustafson, *An Analysis of Diversity in Genetic Programming*. PhD thesis, School of Computer Science and Information Technology, (University of Nottingham, Nottingham, England, 2004, Feb)
41. S. Gustafson, L. Vanneschi, Operator-based distance for genetic programming: Subtree crossover distance. In *Genetic Programming, 8th European Conference, EuroGP2005*, Lecture Notes in Computer Science, LNCS 3447, ed. by M. Keijzer, et al. (Springer, Berlin, Heidelberg, New York, 2005), pp. 178–189
42. S. Gustafson, L. Vanneschi, Operator-based tree distance in genetic programming. IEEE Trans. Evol. Comput. **12**, 4 (2008)
43. J. Hansen, P. Lowry, R. Meservy, D. McDonald, Genetic programming for prevention of cyberterrorism through dynamic and evolving intrusion detection. Decis. Support Syst. **43**(4), 1362–1374
44. E. Hemberg, C. Gilligan, M. O'Neill, A. Brabazon, A grammatical genetic programming approach to modularity in genetic algorithms. In *Proceedings of the 10th European Conference on Genetic Programming (Valencia, Spain, 11–13 Apr 2007)*, vol. 4445 of Lecture Notes in Computer Science, ed. by M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A.I. Esparcia-Alcázar (Springer, 2007), pp. 1–11

45. G. Hornby (2006) ALPS: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, (ACM, New York, NY, USA, 2006), pp. 815–822
46. J. Hu, E. Goodman, K. Seo, Z. Fan, R. Rosenberg, The hierarchical fair competition (hfc) framework for sustainable evolutionary algorithms. Evol. Comput. **13**(2), 241–277 (2005)
47. T. Hu, W. Banzhaf, Neutrality and variability: two sides of evolvability in linear genetic programming. In *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (Montreal, 8–12 July 2009)*, 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, 2009), pp. 963–970
48. 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 (Tuebingen, Apr 15–17 2009)*, vol. 5481 of LNCS, ed. by L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, M. Ebner (Springer, 2009), pp. 85–96
49. E. Jablonka, M. Lamb, *Evolution in Four Dimensions: Genetic, Epigenetic, Behavioral, and Symbolic Variation in the History of Life* (MIT Press, Cambridge, 2005)
50. D. Jakobović, L. Budin, Dynamic scheduling with genetic programming. In *Proceedings of the 9th European Conference on Genetic Programming (Budapest, Hungary, 10–12 Apr. 2006)*, vol. 3905 of Lecture Notes in Computer Science, ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Springer, 2006), pp. 73–84
51. I. Jonyer, A. Himes, Improving modularity in genetic programming using graph-based data mining. In *Proceedings of the 19th International Florida Artificial Intelligence Research Society Conference (Melbourne Beach, FL, USA, May 11–13 2006)*, ed. by G.C.J. Sutcliffe, R.G. Goebel (American Association for Artificial Intelligence, 2006), pp. 556–561
52. W. Kantschik, W. Banzhaf, Linear-tree GP and its comparison with other GP structures. In Genetic Programming, Proceedings of EuroGP'2001 (Lake Como, Italy, 18–20 Apr. 2001), vol. 2038 of LNCS, ed. by J.F. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A.G.B. Tettamanzi, W.B. Langdon (Springer, 2001), pp. 302–312
53. W. Kantschik, W. Banzhaf, Linear-graph GP—a new GP structure. In *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002 (Kinsale, Ireland, 3–5 Apr. 2002)*, vol. 2278 of LNCS, ed. by J.A. Foster, E. Lutton, J. Miller, C. Ryan, A.G.B. Tettamanzi (Springe, 2002), pp. 83–92
54. N. Kashtan, U. Alon, Spontaneous evolution of modularity and network motifs. In *Proceedings of the National Academy of Sciences 102, 39 (27 Sept 2005)*, pp. 13773–13778
55. N. Kashtan, E. Noor, U. Alon, Varying environments can speed up evolution. In *Proceedings of the National Academy of Sciences 104, 34 (21 Aug 2007)*, pp. 13711–13716
56. H. Katirai, Filtering junk E-mail: *A performance comparison between genetic programming and naive bayes*. 4A Year student project, 10 Sept 1999
57. M. Keijzer, V. Babovic, C. Ryan, M. O'Neill, M. Cattolico, Adaptive logic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (San Francisco, California, USA, 7–11 July 2001)*, ed. by L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, E. Burke (Morgan Kaufmann, 2001), pp. 42–49
58. R.E. Keller, R. Poli, Toward subheuristic search. In *Proceedings of 2008 IEEE Congress on Evolutionary Computation* (IEEE Press, 2008) pp. 3147–3154
59. K.E. Kinnear Jr., Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1st IEEE Conference on Evolutionary Computing*, (IEEE Press, Piscataway, NY, 1994), pp. 142–147
60. M. Kirschner, J. Gerhart, J. Norton, *The plausibility of life: Resolving Darwin's dilemma* (Yale Univ Pr, 2006)
61. M. Kotanchek, *The data modeler add-on package for mathematica*. see http://www.evolved-analytics.com/datamodeler (72 2009)
62. J.R. Koza, Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence IJCAI-89 (Detroit, MI, USA, 20–25 Aug 1989)*, vol. 1, ed. by N.S. Sridharan (Morgan Kaufmann, 1989), pp. 768–774

63. J.R. Koza, A genetic approach to the truck backer upper problem and the inter-twined spiral problem. In *Proceedings of IJCNN International Joint Conference on Neural Networks*, vol. IV (IEEE Press, 1992), pp. 310–318
64. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, MA, 1992)
65. J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press, Cambridge MA, 1994)
66. J.R. Koza, D. Andre, F.H. Bennett III, M. Keane, *Genetic Programming 3: Darwinian Invention and Problem Solving* (Morgan Kaufman, San Francisco, CA, 1999)
67. J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence* (Kluwer, Dordrecht, 2003)
68. I. Kushchu, An evaluation of evolutionary generalization in genetic programming. Artif. Intell. Rev. **18**(1), 3–14
69. W. Langdon, A many threaded cuda interpreter for genetic programming. In *Proceedings of the 13th European Conference on Genetic Programming*, vol. LNCS 6021, ed. by A.I. Esparcia-Alcázar, A. Ekárt, S. Silva, S. Dignum, A. Uyar (Springer, 2010), pp. 146–158
70. W. Langdon, W. Banzhaf, Repeated patterns in genetic programming. Nat. Comput. **7**(4), 589–613 (2008)
71. W.B. Langdon, *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, vol. 1 of Genetic Programming (Kluwer, Boston, 1998, Apr 24)
72. W.B. Langdon, W. Banzhaf, Genetic programming bloat without semantics. In *Parallel Problem Solving from Nature—PPSN VI 6th International Conference (Paris, France, 16–20 Sept. 2000)*, vol. 1917 of LNCS, ed. by M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, H.-P. Schwefel (Springer, Berlin, 2000), pp. 201–210
73. W. B. Langdon, W. Banzhaf, Repeated sequences in linear genetic programming genomes. Complex Syst. **15**(4), 285–306 (2005)
74. W. B. Langdon, W. Banzhaf, Repeated patterns in genetic programming. Nat. Comput. **7**(4), 589–613 (2008, Dec)
75. W.B. Langdon, S. Gustafson, J.R. Koza, *GP Bibliography*. http://www.cs.bham.ac.uk/wbl/biblio/gp-bib-info.html (2008)
76. W.B. Langdon, R. Poli, Genetic programming bloat with dynamic fitness. In *Proceedings of the 1st European Workshop on Genetic Programming (Paris, 149-15 Apr 1998)*, vol. 1391 of LNCS, ed. by W. Banzhaf, R. Poli, M. Schoenauer, T. C. Fogarty (Springer, 1998), pp. 96–112
77. W.B. Langdon, R. Poli, *Foundations of Genetic Programming* (Springer, Berlin, 2002)
78. W.-C. Lee, Genetic programming decision tree for bankruptcy prediction. In *Proceedings of the 2006 Joint Conference on Information Sciences, JCIS 2006 (Kaohsiung, Taiwan, ROC, 8–11 Oct 2006)* (Atlantis Press, 2006)
79. S. Luke, ECJ. URL: http://www.cs.gmu.edu/eclab/projects/ecj/ (2010)
80. T. McConaghy, H. Leung, V. Varadan, Functional reconstruction of dynamical systems from time series using genetic programming. In *26th Annual Conference of the IEEE Industrial Electronics Society, IECON 2000 (Nagoya, 22–28 Oct 2000)*, vol. 3, (IEEE, 2000), pp. 2031–2034
81. R.I.B. McKay, N.X. Hoai, P.A. Whigham, Y. Shan, M. O'Neill, Grammar-based genetic programming a survey. Genet. Program. Evolvable Mach. (this issue) (2010)
82. 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 (Naples, 26–28 Mar. 2008), vol. 4971 of Lecture Notes in Computer Science*, ed. by M. O'Neill, L. Vanneschi, S. Gustafson, A.I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, E. Tarantino (Springer, 2008), pp. 134–145
83. J. Merelo, M. Keijzer, M. Schoenauer, *Eo Evolutionary Computation Framework*. URL: http://www.eodev.sourceforge.net/ (2006)
84. 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 1st European Conference on Artificial Life*, ed. by F.J. Varela, P. Bourgine (The MIT Press, Cambridge, 1992), pp. 245–254
85. T. Mitchell, *Machine Learning* (McGraw Hill, New York, 1996)
86. D.J. Montana, Strongly typed genetic programming. Evol. Comput. **3**(2), 199–230 (1995)

87. J. Moore, P. Andrews, N. Barney, B. White, Development and evaluation of an open-ended computational evolution system for the genetic analysis of susceptibility to common human diseases. Lect. Notes Comput. Sci. **4973**, 129–140 (2008)
88. J. Moore, C. Greene, P. Andrews, B. White, Does Complexity Matter? Artificial Evolution, Computational Evolution and the Genetic Analysis of Epistasis in common human Diseases. *Genet. Program. Theory Practice* **VI**, 125 (2008)
89. R. Morrison, *Designing Evolutionary Algorithms for Dynamic Environments* (Springer, Berlin, 2004)
90. Q.U. Nguyen, T.H. Nguyen, X.H. Nguyen, M. O'Neill, Improving the generalisation ability of genetic programming with semantic similarity based crossover. In vol. LNCS 6021, ed. by A.I. Esparcia-Alcázar, A. Ekárt, S. Silva, S. Dignum, A. Uyar (Springer), pp. 184–195
91. Q.U. Nguyen, M. O'Neill, X. H. Nguyen, B. McKay, E.G. Lopez, Semantic similarity based crossover in GP: The case for real-valued function regression. In *Evolution Artificielle, 9th International Conference (26–28 Oct 2009)*, Lecture Notes in Computer Science, ed. by P. Collet, pp. 13–24
92. M. Nicolau, M. Schoenauer, W. Banzhaf, Evolving genes to balance a pole. In vol. LNCS 6021, ed. by A.I. Esparcia-Alcázar, A. Ekárt, S. Silva, S. Dignum, A. Uyar (Springer), pp. 196–207
93. P. Nordin, W. Banzhaf, F.D. Francone, introns in nature and in simulated structure evolution. In *Bio-Computation and Emergent Computation (Skovde, Sweden, 1–2 Sept 1997)*, ed. by D. Lundh, B. Olsson, A. Narayanan (World Scientific Publishing, 1997)
94. M. Oltean, Evolving evolutionary algorithms using linear genetic programming. Evol. Comput. **13**(3)(Fall 2005), 387–410
95. M. O'Neill, A. Brabazon, Recent patents in genetic programming. Recent Pat. Comput. Sci. **2**(1)(2009),43–49
96. M. O'Neill, J. McDermott, J.M. Swafford, J. Byrne, E. Hemberg, E. Shotton, C. McNally, A. Brabazon, M. Hemberg, Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. Int. J. Des. Eng. **3** (2010)
97. M. O'Neill, C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, vol. 4 of Genetic Programming (Kluwer, 2003)
98. U.-M. O'Reilly, M. Hemberg, Integrating generative growth and evolutionary computation for form exploration. In *Genetic Programming and Evolvable Machines 8*, 2 (June 2007), pp. 163–186. Special issue on developmental systems
99. A. Orfila, J.M. Estevez-Tapiador, A. Ribagorda, Evolving high-speed, easy-to-understand network intrusion detection rules with genetic programming. In *Applications of Evolutionary Computing, EvoWorkshops2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoPhD, EvoSTOC, EvoTRANSLOG (Tubingen, Germany, 15–17 Apr 2009)*, ed. by M. Giacobini, I. De Falco, M. Ebner (LNCS, Springer, 2009)
100. P. Domingos. The role of Occam's razor in knowledge discovery. Data Min Knowl Discov **3**(4), 409–425 (1999)
101. R. Poli, M. Graff, (2009) There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009 (Tuebingen, Apr 15–17 2009)*, vol. 5481 of LNCS, ed. by L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, M. Ebner (Springer, 2009), pp. 195–207
102. R. Poli, M. Graff, N.F. McPhee, Free lunches for function and program induction. In *FOGA '09: Proceedings of the 10th ACM SIGEVO Workshop on Foundations of Genetic Algorithms (Orlando, FL, USA, 9–11 Jan 2009)* (ACM, 2009), pp. 183–194
103. R. Poli, W.B. Langdon, O. Holland, Extending particle swarm optimisation via genetic programming. In *Proceedings of the 8th European Conference on Genetic Programming (Lausanne, Switzerland, 30 Mar–1 Apr 2005)*, vol. 3447 of Lecture Notes in Computer Science, ed. by M. Keijzer, A. Tettamanzi, P. Collet, J.I. van Hemert, M. Tomassini (Springer, 2005), pp. 291–300
104. R. Poli, W.B. Langdon, N.F. McPhee, *A Field Guide to Genetic Programming*. Published via http://www.lulu.com and freely available at http://www.gp-field-guide.org.uk (2008). (With contributions by J.R. Koza)
105. R. Poli, N.F. McPhee, Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In *Genetic Programming, Proceedings of EuroGP'2001*, vol. 2038 of LNCS, ed. by J. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A. Tettamanzi, W. Langdon (Springer, 2001), pp. 126–142

106. R. Poli, N.F. McPhee, General schema theory for genetic programming with subtree swapping crossover: Part I. Evol. Comput. **11**(1):53–66
107. R. Poli, N.F. McPhee, General schema theory for genetic programming with subtree swapping crossover: Part II. Evol. Comput. **11**(2):169–206
108. R. Poli, L. Vanneschi, Fitness-proportional negative slope coefficient as a hardness measure for genetic algorithms. In*Genetic and Evolutionary Computation Conference, GECCO'07*, ed. by D. Thierens et al. (ACM Press, 2007), pp. 1335–1342
109. R. Poli, L. Vanneschi, W.B. Langdon, N.F. McPhee, Theoretical results in genetic programming: The next ten years?. Genet. Program. Evolvable Mach. (this issue) (2010)
110. B. Punch, D. Zongker, E. Goodman, (1996) The royal tree problem, a benchmark for single and multiple population genetic programming. In *Advances in Genetic Programming 2*, ed. by P. Angeline, K. Kinnear (The MIT Press, Cambridge, MA, 1996), pp. 299–316
111. J. Rissanen, Modeling by shortest data description. Automatica **14**, 465–471 (1978)
112. J.P. Rosca, Towards automatic discovery of building blocks in genetic programming. In *Working Notes for the AAAI Symposium on Genetic Programming* (AAAI, 1995), pp. 78–85
113. F. Rothlauf, *Representations for genetic and evolutionary algorithms*, 2nd edn. (Springer, pub-SV:adr, 2006). First published 2002, 2nd edition available electronically
114. F. Rothlauf, M. Oetzel, On the locality of grammatical evolution. In Proceedings of the 9th European Conference on Genetic Programming (Budapest, Hungary, 10–12 Apr 2006), vol. 3905 of Lecture Notes in Computer Science, ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Springer, 2006), pp. 320–330
115. C. Ryan, M. Keijzer, An analysis of diversity of constants of genetic programming. In *Genetic Programming, Proceedings of EuroGP'2003 (Essex, 14–16 Apr 2003)*, vol. 2610 of LNCS, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa (Springer, 2003), pp. 404–413
116. G. Seront, External concepts reuse in genetic programming. In *Working Notes for the AAAI Symposium on Genetic Programming (MIT, Cambridge, MA, USA, 10–12 Nov 1995)*, ed. by E.V. Siegel, J.R. Koza (AAAI, 1995), pp. 94–98
117. S. Shekhar, M.B. Amin, Generalization by neural networks. IEEE Trans. Knowl. Data Eng. **4** (1992)
118. 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 (Montreal, 8–12 July 2009)*, 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, 2009), pp. 1115–1122
119. S.G.O. Silva, *GPLab. A Genetic Programming Toolbox for MATLAB, 2008*. See http://www.gplab.sourceforge.net
120. S. Smith, *A learning system based on genetic adaptive algorithms*
121. A.J. Smola, B. Scholkopf. *A Tutorial on Support Vector Regression*. Tech. Rep. Technical Report Series - NC2-TR-1998-030, NeuroCOLT2 (1999)
122. D. Song, M.I. Heywood, A.N. Zincir-Heywood, A linear genetic programming approach to intrusion detection. In *Genetic and Evolutionary Computation—GECCO-2003 (Chicago, 12–16 July 2003)*, vol. 2724 of LNCS, ed. by E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M.A. Potter, A.C. Schultz, K. Dowsland, N. Jonoska, J. Miller (Springer, 2003), pp. 2325–2336
123. L. Spector, Evolving control structures with automatically defined macros. In *Working Notes for the AAAI Symposium on Genetic Programming (MIT, Cambridge, MA, USA, 10–12 Nov 1995)*, ed. by E.V. Siegel, J.R. Koza (AAAI, 1995), pp. 99–105
124. L. Spector, A. Robinson, Genetic programming and autoconstructive evolution with the push programming language. Genet. Program. Evolvable Mach. **3**(1), 7–40 (2002, March)
125. G.F. Spencer, Automatic generation of programs for crawling and walking. In *Advances in Genetic Programming*, ed. by K.E. Kinnear, Jr. (MIT Press, 1994), ch. 15, pp. 335–353
126. P.F. Stadler, Fitness landscapes. In *Biological Evolution and Statistical Physics (Heidelberg, 2002)*, vol. 585 of Lecture Notes Physics, ed. by M. Lässig, Valleriani (Springer, 2002), pp. 187–207
127. P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, S. Tiwari, *Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization*. Tech. Rep. Technical Report Number 2005005, Nanyang Technological University (2005)

128. A. Teller, M. Veloso, PADO: A new learning architecture for object recognition. In *Symbolic Visual Learning*, ed. by K. Ikeuchi, M. Veloso (Oxford University Press, Oxford, 1996), pp. 81–116
129. M. Tomassini, L. Vanneschi, P. Collard, M. Clergue, A study of fitness distance correlation as a difficulty measure in genetic programming. Evol. Comput. **13**(2) (Summer 2005), 213–239
130. L. Vanneschi, *Theory and Practice for Efficient Genetic Programming*. PhD thesis, Faculty of Sciences, University of Lausanne, Switzerland (2004)
131. L. Vanneschi, M. Castelli, S. Silva, Measuring bloat, overfitting and functional complexity in genetic programming. In *GECCO '10: Proceedings of the 12th Annual conference on Genetic and Evolutionary Computation*, ed. by J. Branke (2010)
132. 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 (Montreal, 8–12 July 2009)*, 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, 2009), pp. 1895–1896
133. L. Vanneschi, S. Gustafson, Using crossover based similarity measure to improve genetic programming generalization ability. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (New York, NY, USA, 2009)* (ACM, 2009), pp. 1139–1146
134. L. Vanneschi, S. Gustafson, G. Mauri, Using subtree crossover distance to investigate genetic programming dynamics. In *Genetic Programming, 9th European Conference, EuroGP2006*, Lecture Notes in Computer Science, LNCS 3905, ed. by P. Collet et al. (Springer, Berlin, Heidelberg, New York, 2006), pp. 238–249
135. 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 (London, 7–11 July 2007)*, vol. 2, ed. by D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J.F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R.A. Watson, I. Wegener (ACM Press, 2007), pp. 1759–1759
136. 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 (Budapest, Hungary, 10–12 Apr. 2006)*, vol. 3905 of Lecture Notes in Computer Science, ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Springer, 2006), pp. 178–189
137. E. J. Vladislavleva, G. F. Smits, D. den Hertog, Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans. Evol. Comput. **13**(2), 333–349 (2009, Apr)
138. A. Wagner, *Robustness and Evolvability in Living Systems* (Princeton University Press, Princeton, NJ, 2005)
139. N. Wagner, Z. Michalewicz, M. Khouja, R. McGregor, Time series forecasting for dynamic environments: The dyfor genetic program model. IEEE Trans. Evol. Comput. **11**(4), 433–452 (2006)
140. 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 (Atlanta, GA, USA, 12–16 July 2008)*, 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, 2008), pp. 1315–1322
141. W. Weimer, T. Nguyen, C. Le Gues, S. Forrest, Automatically finding patches using Genetic Programming. In *International Conference on Software Engineering (ICSE) 2009*, (ACM, New York, NY, 2009) pp. 364–374
142. P.A. Whigham, *Grammatical Bias for Evolutionary Learning*. PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 14 Oct 1996
143. P.A. Whigham, Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications (Tahoe City, CA, USA, 9 July 1995)*, ed. by J.P. Rosca, pp. 33–41

144. G. Wilson, M. Heywood, Introducing probabilistic adaptive mapping developmental genetic programming with redundant mappings. Genet. Program. Evolvable Mach. **8**(2), 187–220 (2007, June) Special issue on developmental systems
145. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)
146. J.R. Woodward, Modularity in genetic programming. In *Genetic Programming, Proceedings of EuroGP'2003 (Essex, 14–16 Apr 2003)*, vol. 2610 of LNCS, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa (Springer, 2003), pp. 254–263
147. S. Wright, The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In *Proceedings of the 6th International Congress on Genetics*, vol. 1, ed. by D. Jones (1932), pp. 355–366
148. H. Xie, M. Zhang, P. Andreae, Genetic programming for automatic stress detection in spoken english. In *Applications of Evolutionary Computing, EvoWorkshops2006: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoInteraction, EvoMUSART, EvoSTOC (Budapest, 10–12 Apr 2006)*, vol. 3907 of LNCS, ed. by F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J.H. Moore, J. Romero, G.D. Smith, G. Squillero, H. Takagi (Springer, 2006), pp. 460–471
149. S. Yang, Y.-S. Ong, Y. Jin, Special issue on evolutionary computation in dynamic and uncertain environments. Genet. Program. Evolvable Mach. **7**, 4 (2006)
150. M. Zhang, U. Bhowan, B. Ny, Genetic programming for object detection: A two-phase approach with an improved fitness function. Electron. Lett. Comput. Vis. Image Anal. **6**(1), 27–43 (2006)