Evolutionary Computation and Genetic Programming

Wolfgang Banzhaf

Department of Computer Science, Memorial University of Newfoundland, St. John's, A1B 3X5, CANADA

Abstract

We discuss Evolutionary Computation, in particular Genetic Programming, as examples of drawing inspiration from biological systems. We set the choice of evolution as a source for inspiration in context, discuss the history of Evolutionary Computation and its variants before looking more closely at Genetic Programming. After a discussion of methods and the state-of-the-art, we review application areas of Genetic Programming and its strength in providing human-competitive solutions.

Keywords

Bio-inspired computing, Evolutionary Computation, Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming, Artificial Intelligence, differential evolution, machine learning, automatic programming, natural selection, breeding, population, reproduction, mutation,

crossover, generation, search space, human-competitive.

1 Bio-inspired Computing

One of the more prominent examples of taking bio-inspiration is the application of this philosophy to the development of new ways to organize computation. Life can be paraphrased by describing it as "information processing in a body". Hence biology, the science of the living, has long been concerned with these two key aspects of life: structure and dynamics. The structure of the body, from a single-celled organism like a bacterium to extended, highly complex, multi-cellular, intelligent beings like mammals [1], is a study object of evolutionary, developmental, and molecular biology, among others. Other branches of biology are concerned with the dynamics of behaviour of organisms, in relation to an inanimate environment as well as in relation to other organisms that often provide an important and dynamically changing part of their environment. Ultimately, behaviour requires intense processing of information, both for survival and for the benefit of an organism. Behaviour of individuals is studied in a branch of biology called ethology, the behaviour of species in their interaction with the environment is studied in ecology, while

[1] The spatially largest organism is a fungus, *Armillaria solipides*, with an extension of 8.9 km^2, whereas the largest genome of a vertebrate is that of a fish, *Protopterus aetiopicus*, with a size of 130 Giga bases (as compared to the 3.2 Giga bases of *Homo sapiens*). A base is one of four nucleotides in the alphabet of DNA.

the dynamics of species over time is the subject of evolutionary biology. Molecular biology considers the regulation of behaviour on the molecular level. From the lowest level of molecules to the highest level of evolution of species, this dynamics is about reception and processing of information, and the appropriately executed actions following from the results of such *computation*.

Given this context, it is no wonder that computer scientists and engineers have embraced the paradigms of biology and tried to extract ideas from the living world to apply them in man-made computing environments such as computers and robots. Robots are actually the application area of bio-inspiration closest to actual living organisms, as they can be said to possess a body, a structure that has to act in the real world. Less obvious, yet very active, is the area of bio-inspired computing, where researchers try to extract more or less abstract principles and procedures from living organisms, and realize them in a computational (algorithmic, software) setting.

There is full agreement in the sciences now that the generation of successive sequences of species in what has been called the *tree of life* is a product of evolution, governed by the principles of Darwin's theory of natural selection [1]. Evolution and its models are the source of bio-inspiration that we shall discuss in this chapter in more detail. In a way, this is the most fundamental part of Biology, because it is the driving mechanism for the diversity of life on our planet. However, to set this in context, we want to at least mention in the remainder of this section a number of other examples of bio-inspired

computing, not necessarily in the temporal order of their development.

All aspects of adaptation of organisms to their environment, including the appearance of intelligent behaviour have been used as (bio-) inspiration. The field of *neural networks* [2], for example, has taken inspiration from the structure and function of nervous tissue in higher-order animals, including the brains of humans. The field of *fuzzy logic* [3] took inspiration from human cognitive processes with the ability to think in non-crisp terms. The field of *artificial immune systems* [4] has taken inspiration from the elaborate adaptive systems within higher-order beings that enables them to defend themselves against intruders. The field of *ant colony optimization* [5] has taken inspiration from the distributed nature of ant colonies and their apparently purposeful molding of the environment. The field of *swarm intelligence* [6] has taken inspiration from different sorts of animals organizing their behaviour in swarms, flocks or schools, in order to achieve macro-effects (on the level of the entire swarm) from micro-causes (behaviour of individuals). The field of *artificial life* [7] has taken inspiration from the very beginnings and basics of life, to find ways to produce behaviour akin to living behaviour for the benefit of, e.g., computer games or for the simulation of alternatives to living matter, in order to better understand Life on Earth.

May this list suffice for the moment. It is not exhaustive, and year after year new ideas are being proposed for computation derived from biological systems. Probabilistic reasoning, machine learning, emergence of novelty,

complex adaptive systems, social behaviour, intelligence, sustainability and survival are all terms that can be related to and studied in models of bio-inspired computing. Essential to these models is the idea that a distributed system of interacting entities can bring about effects that are not possible to produce by single entities or entities isolated from each other.

In this chapter let us focus, however, on one particular paradigm within the area of bio-inspired computing, an area which is very intimately connected to all signs of life: evolution. After a general discussion of algorithms derived from evolution (*Evolutionary Algorithms* or *Evolutionary Computing*), we consider in more detail the most modern branch of this area, *Genetic Programming*.

2 History and Variants of Evolutionary Computing

Evolutionary Algorithms or Evolutionary Computing is an area of Computer Science which applies heuristic search principles inspired by natural evolution to a variety of different domains, notably to parameter optimization or other types of problem solving traditionally considered in Artificial Intelligence.

Early ideas in this field developed at a time when computers were barely commercially sold. Alan Turing was one of the first authors to correctly identify the power of evolution for the purpose of solving problems and exhibiting

intelligent behaviour. In his 1950 essay "Computing Machinery and Intelligence" [8] Turing considered the question whether machines could think. He was concerned that digital computers - despite their power and universality - would only be capable of executing programs deterministically. He felt that this would not be sufficient to produce intelligent behaviour. "Intelligent behaviour presumably consists in a departure from the completely disciplined behaviour involved in computation, but a rather slight one, which does not give rise to random behaviour ..." ([8, p. 457]). Computation here refers to the only known form of digital computation at the time: deterministic computation. Turing pointed out that a digital computer with a random element would be an interesting variant of such a machine, especially, "when we are searching for a solution of some problem." It was clear to Turing that machines at some point would need to be able to learn in a fashion similar to children - a very clear indication of him taking inspiration from biology. "Now the learning process may be regarded as a search [...]. Since there is probably a large number of satisfactory solutions, the random method seems to be better than the systematic. It should be noticed that it is used in the analogous process of evolution." [8, p. 459]

So, already in 1950 several ideas were voiced that would lead the way to evolutionary algorithms. The notion of a soft kind of randomness, which would later become mutation and crossover, the notion of intelligent behaviour as the goal of these algorithms, the notion of a search process to achieve learning

and problem solving, and the notion of kinship to evolution in Nature were all entertained in this article by Alan Turing already.

In 1962, a budding computer scientist [2] from the University of Michigan published a paper entitled "Outline for a Logical Theory of Adaptive Systems" in which he proposed most of what later became known as *Genetic Algorithms* [9]. Holland wrote: "The study of adaptation involves the study of both the adaptive system and its environment", thus foreshadowing the necessity to define a fitness function as a stand-in for the environment. He then proposed to look at the adaptive system as a "population of programs", and emphasized the advantage of looking at adaptation from the viewpoint of a population: "There is in fact a gain in generality if the generation procedure operates in parallel fashion, producing sets or populations of programs at each moment rather than individuals."[9, p. 298] Here, Holland correctly identified the strength that populations of solutions bring to a problem, when applied and tested in parallel. "The generated population of programs will act upon a population of problems (the environment) in an attempt to produce solutions. For adaptation to take place the adaptive system must at least be able to compare generation procedures as to their efficiency in producing solutions." What Holland called generation procedures were later termed - in the context of su-

[2] Computer Science as a discipline did not yet even exist. It is only in hindsight that we call it like that; officially, it was called "Communication Science" at the time.

pervisory programs, that is programs that allow a system to adapt to different environmental conditions - mutation. After introducing differential selection as a key driving force for adaptation "Adaptation, then, is based upon differential selection of supervisory programs. That is, the more "successful" a supervisory program, in terms of the ability of its problem-solving programs to produce solutions, the more predominant it is to become (in numbers) in a population of supervisory programs. [...] Operation of the selection principle depends upon continued generation of new varieties of supervisory programs. There exist several interesting possibilities for producing this variation."[9, p. 300/301] "The procedure described [...] requires that the supervisory program duplicate, with some probability of variation or mutation [...]." [9, p. 309]

Thus, as early as 1962, a sketch of evolutionary algorithms was in place, that could only become more pronounced over the years [10]. The randomness of Turing's paper, later still reflected in Friedberg's work [11], gave way to a variation-selection loop, with accumulation of beneficial variations in a population and the regular information exchange between individuals.

Other paradigmatic developments in evolutionary algorithms at the time include *Evolutionary Programming* [12], and *Evolutionary Strategies* [13]. For a more thorough review of early work in Evolutionary Computing the reader is pointed to Ref. [14], discussing a selection of papers from the "fossil record" of evolutionary computing.

All Evolutionary Algorithms follow the Darwinian principle of differential natural selection. This principle states that the following preconditions must be fulfilled for evolution to occur via (natural) selection:

a) There are entities called individuals which form a population. These entities can reproduce or can be reproduced.
b) There is heredity in reproduction, that is to say that individuals of this population produce similar offspring.
c) In the course of reproduction there is variety which affects the likelihood of survival and, therefore, the reproducibility of individuals. This variety is produced by stochastic effects (like random mutation and recombination) as well as by systematic effects (like mating of like with alike, etc.).
d) There are finite resources which cause the individuals to compete. Due to overreproduction of individuals, not all can survive the struggle for existence. Differential natural selection is a result of this competition exerting a continuous pressure towards adapted or improved individuals relative to the demands of their environment.

In Evolutionary Algorithms, there is a population of individual solutions. Usually, this population is initialized as a random population, i.e., from random elements determined to be potentially useful in this environment. Next is a determination of fitness in the process of evaluation. This could take any form of a measurement or calculation to determine the relative strength of an individual solution. The outcome of this measurement or calculation is then

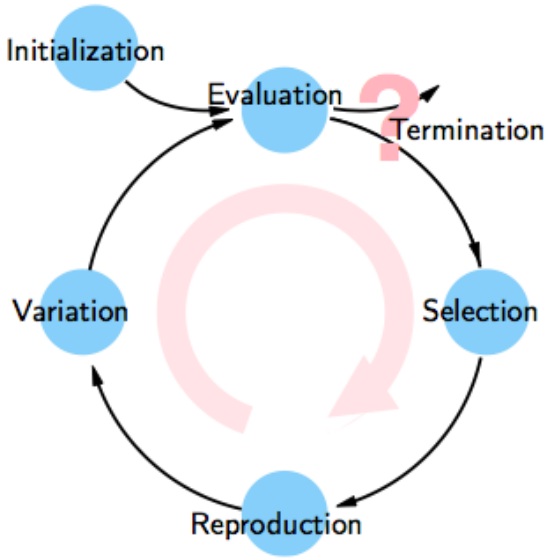


Figure 1

The general process of an evolutionary algorithm: A cycle of evaluation, selection and variation that accumulates beneficial changes. Variation operators could be mutation, duplication or crossover/recombination.

used in the selection step to determine which individual solutions are to survive the competition for resources, and which are to be replaced by copies or variants of the surviving individual solutions. The last step is to apply variation to these individual solutions so that the population is complete again and ready for the next round of evaluation. Figure 1 summarizes the general process.

Now that we have discussed the general approach, it is time to turn our attention to Genetic Programming.

3 Genetic Programming - History, Principles, and Methods

While John Holland became known for the Genetic Algorithm [10], he already had spoken of programs in his seminal 1962 paper. It was, however, the long held mainstream view in Computer Science, that subjecting computer programs to the random forces of mutation and recombination would not yield viable programs. The notion was that computer code is too brittle to be improved by randomness recruited in evolution.

The term *Genetic Programming (GP)* [15] describes a research area within the field of Evolutionary Computation that deals with the evolution of computer code. Its algorithms aim either to approximate solutions to problems in machine learning or to induce precise solutions in the form of grammatically correct (language) structures for the automatic programming of computers. Again, the same general process as depicted in Figure 1 is applied, but this time to structures that determine the behaviour of a computer.

It took a long time to realize that it is not impossible to evolve computer code. This exciting development needed a number of different (smaller) intermediate steps, starting from the original Genetic Algorithms using fixed-length bit strings to represent numbers in optimization problems. The first realization was that rule systems, instead of numbers representing problem solutions, could be subjected to evolution. In another seminal paper, Holland and Reitman introduced the *classifier system* [16] which allowed *if-then*-rules

under evolution. Classifier systems have since taken on a life of their own [17], but the key contribution to the future field of Genetic Programming was that *if-then*-rules are a hallmark of programming languages and execution of more complex computer code. However, a classifier system cannot be regarded as a system for evolutionary program induction, because there are no program individuals under evolution. [3]

Not long after Holland and Reitman, Smith introduced a variable-length representation allowing it to concatenate rules into rule-based *programs* which can solve a task defined by a fitness function [18]. In 1981, Forsyth [19] published a logical rule system with parameters that allows us to classify examples into different classes. Here, a training set of data samples can be used to train the classifier. The *programs* would logically and numerically evaluate data samples to conclude on the class of the example. Forsyth summarized in wonderful prose: "I see three justifications for this kind of exercise. Firstly, it is interesting in its own right; secondly, the rules behave in an interesting fashion; and thirdly, it seems to work. In the first place it is fun to try a little abstract gardening, growing an orchard of binary trees. And it might be fruitful in another sense. After all, we are only here by courtesy of the principle of natural selection, AI workers included, and since it is so powerful in producing natural intelligence it behooves us to consider it as a method for cultivating the artificial variety." [19, p.163/164] One cannot escape the impression that the

[3] Note that Holland's first paper [9] already discussed programs!

author of these lines couldn't at first believe that his method was so effective at classifying samples.

In the second half of the 1980s, the number of early GP systems proliferated. Cramer introduced in 1985 two evolutionary programming systems based on different simple languages he designed [20]. Hicklin, and Fujiki & Dickinson wrote precursor systems for particular applications, using the standard programming language LISP [21, 22] before Koza in 1989 finally documented a method that both used a universal language and was applied to many different problems [23]. Genetic Programming came into its own with the publication of John Koza's book in 1992 [15]. It is his achievement to have recognized the power and generality of this method, and to document with numerous examples, how the approach can be used in different application areas. In the introduction to his book he wrote: "In particular, I describe a single, unified, domain-independent approach to the problem of program induction - namely genetic programming."

After going through the gradual development of ideas, it is now time to discuss the principles of Genetic Programming. GP works with a population of computer programs that are executed or interpreted in order to judge their behavior. Usually, fitness measurements sample the behavioural space of a program to determine the degree to which the outcome of the behaviour of a program individual is what it is intended for. For instance, the deviation between the quantitative output of a program and its target value (defined

through an error function) could be used to judge the behaviour of the program. This is a straight-forward procedure if the function of the target program can be clearly defined. Results may also be defined as side-effects of a program, such as consequences of the physical behavior of a robot controlled by a genetically developed program. Sometimes, an explicit fitness measure is missing altogether, for instance in a game situation, and the results of the game (winning or loosing) are taken to be sufficient scoring for the program's strategy. Again, very much following the original intuition of Holland, a variety of programs is applied to the same problem and their performances relative to each other are used to determine which programs are to be conserved for future generations, and which ones are to be discarded.

The outcomes of fitness evaluation are used to select programs. There are several different methods for selection, both deterministic and stochastic. Selection determines (a) which programs are allowed to survive (overproduction selection), and (b) which programs are allowed to reproduce (mating selection). Once a set of programs has been selected for further reproduction, the following operators are applied:

- reproduction
- mutation
- crossover

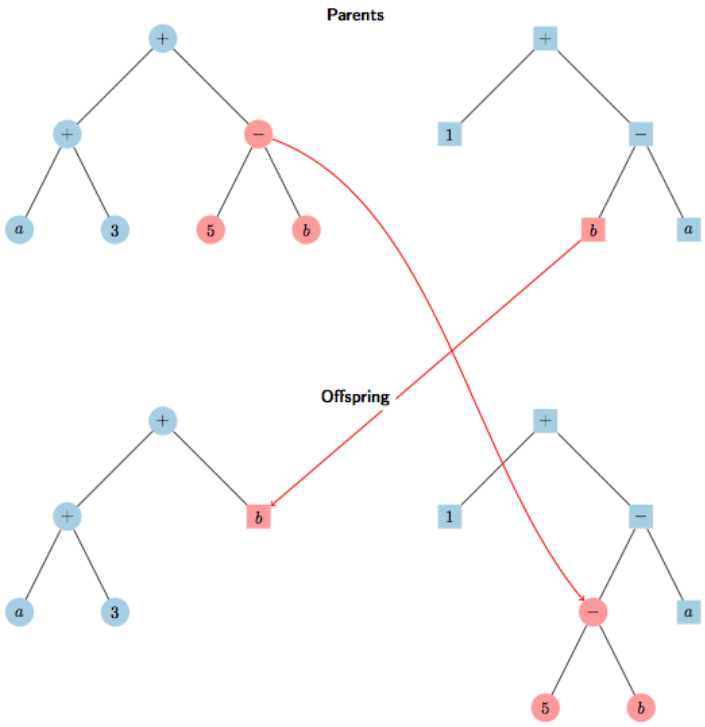
Reproduction simply copies an individual to an offspring population (the next

generation), *mutation* varies the structure of an individual under control of a random number generator, and *crossover* or *recombination* mixes the structures of two (or more) programs to generate one or more new programs for the offspring population. Additional variation operators are applied in different applications. Most of these contain knowledge in the form of heuristic search recipes which are adapted to the problem domain.

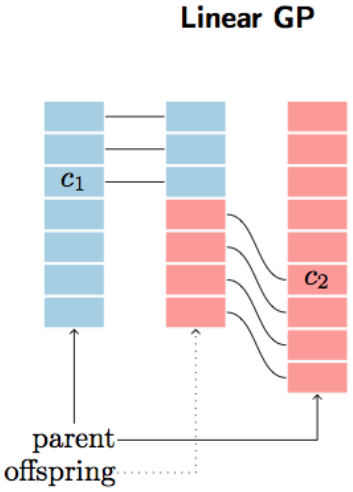
The material under evolution is, as we said, computer code. However, the representation of this code and the implementation of variation operators is important in order to avoid the brittleness of the code.[4] The two most popular representations for computer programs under evolution nowadays are expression trees of functional programming languages and (linear) sequences of instructions from imperative programming languages [24]. Figure 2 shows how these two representations can be subjected to a crossover or recombination operation. There are many other representations for GP, notably those that make use of developmental or generative processes that grow programs, see for instance, Refs. [25, 26]

Once programs have been generated, they are interpreted or compiled to produce behaviour, which is subsequently measured in terms of its fitness. In this way, fitness advantages of individual programs are exploited in a popula-

[4] *Brittleness* of computer code refers to the fact that it can be easily broken, even with the slightest (possibly random) variation, with the result that it doesn't work at all.



(a) Exchanging subtrees.



(b) Exchanging blocks of code.

Figure 2

Tree-based and sequence-based representation of programs and their respective recombination.

tion to lead to better solutions. In Genetic Programming, as well as in other Evolutionary Algorithms, differential selection can be realized in a number of different ways.

The simplest approach is a generational selection scheme called fitness-proportional or roulette-wheel selection. The fitness of all individuals in the population is summed up. The fitness f_i of each individual i is then normalized by the sum of all fitness values found, and this normalized value determines the probability p_i of the individual i of being selected for reproduction/mutation/crossover. Thus, $p_i = f_i / \sum_j f_j$. Based on the stochasticity of random events, fitness proportional selection also allows weak individuals to succeed in reproduction some of the time.

Another selection method is called tournament selection. A subset of individuals of the population is drawn randomly, and are compared to each other in fitness. The individuals with higher fitness are allowed to replace (directly as in reproduction, or with a variation as in mutation and crossover) the individuals with lower fitness. Normally, tournament selection is done with a minimum number of $k = 4$ individuals, which carries the lowest selection pressure. Larger tournaments, however, have been used in the literature, up to the extreme of holding tournaments among the whole population at which point the selection scheme is called truncation selection.

Ranking selection is another selection scheme introduced to address some

weaknesses of fitness proportional selection. Each individual is assigned a rank in the population, and selection proceeds by selecting either by a linearly or exponentially associated probability based on the rank of an individual. A detailed comparison of different selection schemes is given in Ref. [27].

The entire process of selection can be seen in close analogy to breeding animals. The breeder has to select those individuals from the population which carry the targeted traits to a higher degree than others. [5]

All selection schemes rely on a sufficiently accurate determination of fitness to work properly. Therefore, one of the most important ingredients in Genetic Programming is the definition of a fitness measure to determine the appropriateness of a program individual's behaviour. Sometimes the fitness measure has to be iteratively improved in order for the evolved solutions to actually perform the function they were intended for.

Fitness calculation is actually one of the areas where Genetic Programming and other Evolutionary Algorithms differ. GP has to judge the behaviour of a program, a structure that determines the behaviour of a computer under different inputs, i.e., an active entity. This executed program has to produce outputs that adhere as closely as possible to a prescribed behaviour. Thus, while a GA (or another optimization technique) is used to optimize a partic-

[5] Darwin's original theory of evolution was inspired by breeding animals, especially pigeons and cattle.

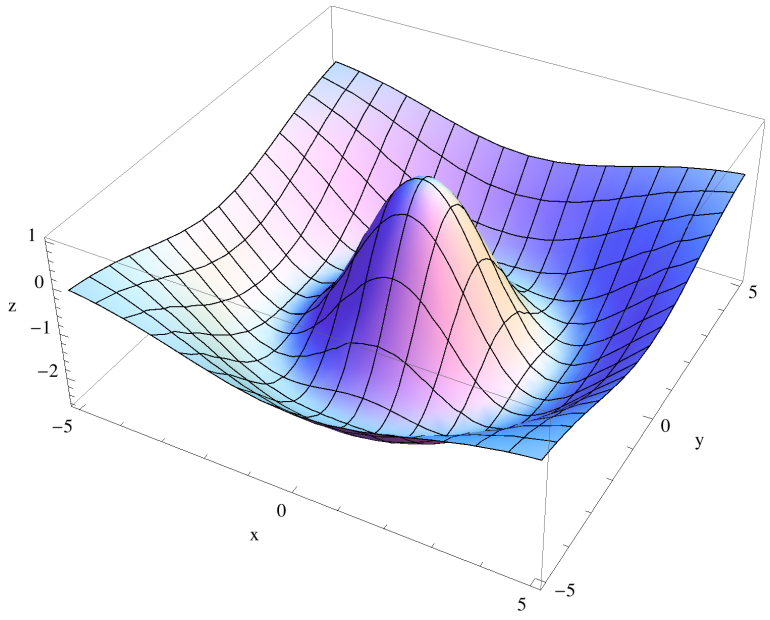


Figure 3

Finding the highest point in a fitness landscape of a function $z = z(x, y)$: A typical GA task.

ular instance of an optimization problem, Genetic Programming has to find behaviour in a larger search space, providing the correct behaviour under a multitude of input/output pairs. The situation can be depicted by considering the difference between (i) optimizing a function, i.e. finding the minimum or maximum of a function as in finding the center ($x = y = 0$) as the function maximum in Figure 3 in the GA task, and (ii) constructing the function whose samples are provided by the data points in Figure 4 in a typical GP task.

Fitness measurement is thus a more extended affair in Genetic Programming, and cannot usually be completed with one sample measurement in the behavioural space. Instead, multiple instances of input/output pairs are presented to the program population, and the results for all pairs are usually

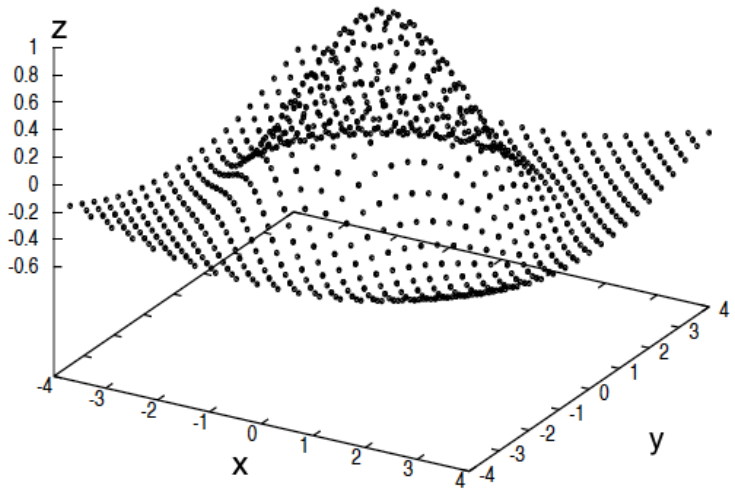


Figure 4

Finding the function $z = z(x, y)$ fitting the data points: A typical GP task.

averaged to arrive at a fitness value for programs which subsequently forms the basis for selection.

A few more words on the difference between GA tasks and GP tasks are in order. One might argue that the GP task is nothing more than an optimization task, looking to minimize the error of the function approximation for Figure 4. If one were to have a fixed-sized genome, consisting of, say coefficients of a polynomial in x and y, assuming an expression of the type

$$z = \sum_{i,j=0}^{n} a_{ij} x^i y^j \tag{1}$$

with a_{ij} being the genetic variables(alleles) that are subject to a GA, this can be legitimately considered a GA. Note, however, that the functional dependencies need to be determined beforehand, both in terms of the dimensionality as well as in terms of the order (in this case, dependency on x and y with

terms up to order n). Some of these terms might still be close to $a_{ij} = 0$, which would indicate either no or only a minute contribution of the term to the overall outcome z.

Conversely, though, one could argue that searching for the maximum in Figure 3 could be construed as a GP task. This could be done by assuming a growing set of fitness cases through probing the landscape for the maximum, based on a model of the landscape. By then trying to find a symbolic expression for the function, based on the existing points probed, calculating its first- and second-order derivatives, and solving for appropriate conditions to find the maximum, a new prediction could be made. This new point would be visited, but with an initially inaccurate model for the function it can be assumed that it would be off somewhat from the real maximum of the function. By using this kind of modeling approach (where GP is in charge of developing and refining the model), it can be easily imagined that such a process could yield a faster approach to the maximum than just sampling the space.

The difference between a GA approach and a GP approach to optimizing fitness of respective solutions is thus best tied to the representation used for the task. Is it a length-changing representation (like in the foregoing example, the expression for a function), or is it a fixed-length representation like in Eq. (1)? Does it use multiple fitness cases or is there only one fitness measurement? The former is a GP approach, the latter a GA approach. Naturally, a GP approach will search a larger space of possibilities, because the combinatorics

Table 1

Comparison of GA vs GP

	GA	GP
Representation	fixed-length	variable-length
Individual	passive	active
Genome	parameters	program/algorithm
Input	none	input values
Fitness	one value	many fitness cases
Typical application	function optimization	function approximation
Typical size of search space	10^{100}	$10^{100,000}$

of its structures is much larger, with typical GP search spaces $10^{1,000}$ times larger than typical GA search spaces. Table 1 summarizes the issues.

4 Advances and State-of-the-Art

In his seminal work of 1992, Koza established the field of GP by arguing convincingly that manipulation of structures of symbolic expressions is possible with evolutionary algorithms and that the resulting technique would have a wide variety of applications. In subsequent years, the field experienced both broadening and deepening [24]. Many different representations for GP

were studied, among them other generic data structures such as sequences of instructions or directed graphs, as well as more exotic data structures such as memory stacks or neural networks. Today, different approaches are considered as GP, from the evolution of expression trees to the evolution of electronic circuits or even architectural designs (*structures*, for short). The overarching principle is to subject all these kinds of structures with variable complexity to forces of evolution by applying mutation, crossover and fitness-based selection. The results must not necessarily be programs, but could be descriptions for designs (like structures of bridges) or other manipulatable elements.

An ever present difficulty with GP is that the evolution of structures of variable complexity (e.g., program code) often leads to individuals with a large number of elements, often with considerable redundancy. Notably it was found that variable complexity often leads to inefficient code that requires a lot of memory space. Several researchers subsequently observed that the evolutionary forces seem to exert a pressure toward more complex solutions, parts of which could be removed after evolution without doing any harm to the behavior of the evolved solution. By drawing an analogy from biological evolution of genomes, this phenomenon was originally called "code bloat", "intron growth", or growth of ineffective code [28]. It was found that code growth is not the only unintended result of evolutionary processes, but it has been the most examined emergent phenomenon to date [29]. [6] At least

[6] Another emergent phenomenon in Genetic Programming is the emergence of

three different influences are at work promoting the growth of complexity during evolution. The most important influence has to do with the protection effect of redundant code if subjected to the action of crossover or mutation. Redundant code is more resistant to crossover and mutation and allows its carrier solution to survive better, compared to other individuals which do not possess this redundancy [31]. Removal bias in crossover operations [32] which describes the fact that code can grow to infinity from any size, but only be reduced to zero from a particular size is another explanation. Finally, a genetic drift toward larger solutions [33] has been named as an important influence.

Over the last decade, the relation between robustness of organisms and their evolvability has been under intense study in biology (see for example Ref. [34]). A seeming paradox between these two features frequently found in Nature has been resolved. It was found that neutral evolution is a key aspect of robustness. Neutral evolution refers to the capability of evolution to change genotypes without changes to phenotypes of individuals.[7] This capability has been known for decades already, and had previously been discovered to be an important process in evolution [35]. The understanding of neutrality led to new mathematical models like the idea of neutral networks [36]. These ideas are beginning to exert influence in the EC community [37] and it turns

repetitive code [30].

[7] The genotype of an individual is its genetic make-up, potentially subjected to mutation and crossover, whereas the phenotype of an individual is the resulting program (behaviour).

out that also in GP solutions that are more robust are preferred through the evolutionary process, another emergent phenomenon [38, 39].

While the GA theory has been well established, theoretical progress in GP has been more difficult to achieve since GP works with variable complexity and multiple fitness cases for fitness scoring. Many researchers are working to produce results for GP by gleaning from GA research. The schema theory of Genetic Algorithms [40, 41] has been a primary target of knowledge transfer. [8] In the meantime, several different schema theorems have been formulated for GP and theory has progressed substantially [42].

When analyzing search spaces of programs, it was realized that their size is many orders of magnitude larger than search spaces of combinatorial optimization problems. A typical size for a program search space might be $10^{100,000}$, as opposed to a typical search space for a combinatorial optimization problem being of the order of 10^{100}. Although this might be interpreted as discouraging for search mechanisms, it was also realized that the solution density in program spaces is, above a certain threshold of complexity, constant with changing complexity [43]. In other words, there are proportionally many more valid solutions in program spaces than in the spaces of combinatiorial

[8] A *schema* in a GA is a sub-pattern of the genotype that takes the form of a template and can be used to identify several genotypes. For instance, in a binary string genotype with 4 bits: $1 * 0*$, all genotypes with a 1 in position 1 and a 0 in position 3 belong to this schema.

optimization problems.

While GP has made great strides over the last two decades, many issues are still open and require continued investigation. Theory of GP [42]

- has succeeded in finding appropriate schema theorems that allow to understand how the search space and the population representation interact,
- has started to analyze Markov chain models of the search process dynamics, and
- has found ways to characterize search spaces (difficulty, complexity) and relate them to the performance of GP systems.

In coming years, GP theory is expected to make progress on the treatment of dynamical problems, proofs of convergence of the search algorithms, and in classifying problem spaces.

On the practical side, GP research will target [44]:

- identifying appropriate representations for Genetic Programming in particular problems,
- the design of open-ended evolutionary systems with GP,
- the problem of generalization in GP,
- the establishment of benchmarks for measuring and comparing GP performance, and
- modularity and scalability in GP.

There is also more room for adding bio-inspiration to Genetic Programming. For instance, the relation between evolution and development has been studied for decades in biology [45]. It was found that the time-dependent process of gene expression and gene regulation through both internal and external cues is the mechanism by which both processes can be unified [46]. Some progress has also been made in Genetic Programming to couple evolution and development. The developmental approach in GP takes the form of a recipe that, upon its execution, generates a structure that is subjected to fitness tests [47]. Thus, it is not the GP program itself that is tested, but the result of its execution.

Similar to the coupling between evolution and development, the coupling between development and learning was considered an important link for understanding the mechanisms of development and learning processes. Cognitive neuroscience has presented evidence for this coupling by finding that there are critical periods in development in which certain learning tasks are facilitated (and sometimes only possible) to take place. If the critical period is missed, learning success in a task is substantially reduced [48].

The coupling between development (or evolution) and learning has only recently been explored in GP. The problem is to clearly separate adaptations or fitness gains resulting from development or evolution versus those from learning. Its reason is the less stringent separation of time scales between evolution, development, and learning in GP systems. While biological evolution can hap-

pen over many thousands or millions of years, development over the lifetime of an organism, and learning over phases of that lifetime, all three mechanisms are on similar time scales in GP, usually tied into single runs of a GP system. In addition, in most GP systems there is no notion of species (and their evolution). Rather, the entire population is essentially mixed and therefore belongs to one single species. Finally, the goal under the influence of evolution is behavior, the same entity usually associated with learning. First attempts to examine learning as a separate task for which evolution/development have to provide the means have been made [49], yet this area requires much more investigation.

5 Applications

The textbook of Banzhaf et al. from 1998 lists 173 GP applications from A to Z already at the time [24]. Fifteen years have passed, and the field has continued to develop rapidly. The main application areas of GP are (from narrow to wide) [24]:

- computer science,
- science,
- engineering,
- business and finance, and
- art and entertainment.

Koza has contributed many interesting applications to some of these areas [50], demonstrating the breadth of the method. However, a more detailed look is warranted.

In Computer Science, much effort has gone into the development of algorithms using GP. By being able to manipulate symbolic structures, GP is one of the few heuristic search methods for algorithms. Sorting algorithms, caching algorithms, compression algorithms [51], random number generators and algorithms for automatic parallelization of code [52], to name a few, have been studied. The spectrum of applications in Computer Science spans from the generation of proofs for predicate calculus to the evolution of machine code for accelerating function evaluation. The general tendency is to try to automate the design process for algorithms of different kinds. Recently the process of debugging code, i.e. the correction of errors has been added to the list of applications [53]. Computer Science itself has many applications, and it is natural that those areas also benefit indirectly by improving methods in Computer Science. For instance, in the area of Computer Vision, Genetic Programming has been used, among others, for

- object detection (for example Refs. [54, 55]),
- filter evolution (for example [56, 57]),
- edge detection (for example [58]),
- interest point detection (for example [59]), and
- texture segmentation (for example [60]).

As well, the area of Software Engineering is a field very fruitful for applications of GP [61]. Query optimization for database applications is a widespread application of evolutionary computation techniques (see their use in PostgreSQL and H2 [62, 63]).

Typical applications for Genetic Programming in Science are those to modeling and pattern recognition. Modeling certain processes in Physics and Chemistry with the unconventional help of evolutionary creativity supports research and understanding of the systems under study [64, 65]. For instance, parameters of models in Soil Science can be readily estimated by GP [66]. Predictions based on models generated with GP have widespread applications. An example from Climate Science is [67], where sea water level is forecast by a Genetic Programming modelling technique using past time series. Many modelling applications for EC methods in general exist in Astronomy and Astrophysics; see for instance Refs. [68, 69].

Modelling is, however, but one of the applications of Genetic Programming in Science. Pattern recognition is another key application, used in molecular biology and other branches of biology and medicine, as well as in Science in general [70, 71]. Here, GP has delivered results that are competitive if not better than human-generated results [72, 73], a special area of applications we have to come back to in the next section. Classification and data-mining are other applications where GP is in prominent use [74, 75].

In Engineering, GP and other evolutionary algorithms are used as stand-alone tools [76] or sometimes in competition or cooperation with other heuristic methods such as Neural Networks or Fuzzy Systems. The general goal is again to model processes such as material properties [77], production plants, or to classify results of production. In recent years, design in Engineering has regained some prominence [78]. Control of man-made apparatus is another area where GP has been used successfully, with process control and robot control (e.g., [79]) the primary applications.

In Business and Finance, GP has been used to predict financial data, notably bancruptcy of companies [80, 81]. The entire area of computational finance is ripe with applications for GP (and other evolutionary techniques), see Refs. [82, 83]. For an early bibliography of business applications of GP and GAs, the reader is referred to Ref. [84]. Since, generally speaking, modelling and prediction are core applications in economic contexts, GP is an important nonlinear modelling technique to consider, see e.g., Refs. [85, 86].

In Art and Entertainment, GP is used to evolve realistic animation scenes and appealing visual graphics (see [87] for an early example). Computer Games are another active area of research and application for Genetic Programming (see for instance [88]). Board games have been studied with GP developed strategies, too [89, 90]. It also has been used in visual art and music [91]. In music, for example, GP was used to extract structural information from musical composition in order to model the process so that automatic composition

of music pieces becomes possible [92].

Many of these problems require a huge amount of computational power on the part of the GP systems. Parallel evolution has hence been a key engineering aspect of developments in GP. As a paradigm, GP is very well suited for a natural way of parallelization. With the advent of inexpensive parallel hardware, in recent years in particular through Graphics Processing Units [93, 94, 95], a considerable proliferation of results is expected from GP systems [96].

6 Human-Competitive Results of Genetic Programming

In the last decade, a substantial number of results have been published in various fields that claim to have produced human-competitive results by the application of GP as a problem solving method [97].

These claims are based on a comparison between the presently-best known human solutions to a problem and their respective counterparts produced by Genetic Programming. Applications are from areas like quantum computing algorithms, analog electrical circuit design and other mechanical and electrical designs, game playing applications, finite algebras and other mathematical systems, bioinformatics and other scientific pattern recognition problems, reverse engineering of systems, and empirical model discovery.

The claims of human-competitiveness are based on criteria that Koza et

al proposed in 2003:

a) The result was patented as an invention in the past, is an improvement over a patented invention, or would qualitfy today as a patentable new invention.

b) The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.

c) The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.

d) The result is publishable in its own right as a new scientific result, independent of the fact that the result was mechanically created.

e) The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.

f) The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.

g) The result solves a problem of indisputable difficulty in its field.

h) The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

Some of the similarities of these successes have been summarized by

Koza [73] as follows:

- Usually, a large amount of computational power has to be invested in order to gain human-competitive results from Genetic Programming runs.
- Most times, a dedicated representation for the solution, known to be efficient by the specialist, has been applied to allow the full power of expression of solutions to be born on the problem.
- The Genetic Programming system has been equipped with dedicated growth or development operators such that the adaptation of complexity of a description can be achieved smoothly.

Due to the ability of the human mind to quickly grasp the recipes of a problem solution that an artificial system has applied, the question remains open whether solutions found by a Genetic Programming system will remain qualitatively better than solutions discovered by human over the long term. Perhaps the best area to consider for this kind of attempt is Mathematics. First results have been achieved that seem to indicate that, under very special circumstances, certain mathematical problems can be solved more efficiently using GP [98].

7 Conclusions

Implementation of GP will continue to benefit in coming years from new approaches including results from developmental biology and epigenetics.

Application of GP will continue to broaden. Many applications focus on engineering applications. In this role, Genetic Programming may contribute considerably to creative solutions to long-held problems in the real world.

Since GP was first used around 1990, raw computational power has increased by roughly a factor of 40,000 following Moore's law of doubling of transistor density every 18 months. As Koza points out, while initially only toy problems were amenable to solution through GP, subsequent increases in computational power and methodological progress of GP has allowed new solutions to previously patented inventions as well as, more recently, completely new inventions that are by themselves patentable. A milestone in this regard was reached in 2005 when the first patent was issued for an invention produced by a Genetic Programming system [99].

The use of bio-inspiration, notably through lessons from our understanding of natural evolution has led to some very substantial progress in the implementation of artificial systems that show human-level problem solving abilities. While achieving "Artficial Intelligence" in computing machines is still far in the future, in restricted areas steps in that direction have been taken. It is the firm conviction of the author of this chapter that a major component of any future system that could truely lay claim to the property of intelligence will be bio-inspiration.

Acknowledgement

I would like to express my sincere gratitude to my students, collaborators and colleagues with whom it is such a pleasure to work on various projects in Evolutionary Computation. I also would like to acknowledge funding agencies that financed many projects over the course of nearly two decades. Specifically I want to mention the German Science Foundation (DFG), the Technical University of Dortmund and the state of Northrhine-Westphalia for funding from 1993 to 2003 and NSERC (Canada), Memorial University of Newfoundland and the government of Newfoundland for funding from 2003 to present.

Bibliography

[1] C Darwin. *On the Origin of Species*. John Murray, London, 1859.

[2] C.M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, UK, 1995.

[3] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic*. Prentice-Hall, New Jersey, 1995.

[4] L.N. De Castro and J. Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, Heidelberg, 2002.

[5] M. Dorigo and C. Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344:243–278, 2005.

[6] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from*

natural to artificial systems. Oxford University Press, New York, USA, 1999.

[7] C. Adami. *Introduction to Artificial Life*. Telos, Springer, New York, 1998.

[8] A.M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[9] JH Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9:297 – 314, 1962.

[10] J Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[11] RM Friedberg. A Learning Machine: Part I. *IBM Journal of Research and Development*, 2:2 – 13, 1958.

[12] L Fogel, A Owens, and M Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[13] I Rechenberg. *Evolution Strategy*. Holzmann Froboog, Stuttgart, Germany, 1974.

[14] D.B. Fogel, editor. *Evolutionary Computation - The Fossil Record*. IEEE Press, New York, 1998.

[15] J Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.

[16] JH Holland and J Reitman. Cognitive systems based on adaptive algorithms. In DA Waterman and F Hayes-Roth, editors, *Pattern Directed Inference Systems*, pages 313 – 329, New York, 1978. Academic Press.

[17] R.J. Urbanowicz and J.H. Moore. Learning classifier systems: a complete

introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009:736398, 2009.

[18] SF Smith. *A Learning System based on Genetic Adaptative Algorithms*. PhD thesis, University of Pittsburgh, 1980.

[19] R Forsyth. BEAGLE - A Darwinian Approach to Pattern Recognition. *Kybernetes*, 10:159 – 166, 1981.

[20] N Cramer. A representation for the adaptive generation of simple sequential programs. In *Proc. First International Conference on Genetic Algorithms*, pages 183 – 187, Pittsburgh, 1985. Lawrence Erlbaum.

[21] JF Hicklin. Application of the genetic algorithm to automatic program generation. Master's thesis, University of Idaho, 1986.

[22] C Fujiki and J Dickinson. Using the genetic algorithm to generate lisp source code to solve the prisoner's dilemma. In *Proc. Second International Conference on Genetic Algorithms*, pages 236 – 240, Pittsburgh, 1987. Lawrence Erlbaum.

[23] J Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *Proc. 11th International Joint Conference on Artificial Intelligence*, pages 768 – 774, San Mateo, CA, 1989. Morgan Kaufmann.

[24] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming - An Introduction*. Morgan Kaufmann, San Francisco, 1998.

[25] C. Ryan, JJ Collins, and M. Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W Banzhaf, R Poli, M Schoenauer,

and T Fogarty, editors, *Proc. EuroGP 1998*, pages 83 – 96, Heidelberg, 1998. Springer.

[26] J.F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In W. Banzhaf et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pages 1135–1142, San Francisco, 1999. Morgan Kaufmann.

[27] T Blickle and L Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4:361–394, 1996.

[28] P Angeline. Genetic programming and emergent intelligence. In KE Kinnear, editor, *Advances in Genetic Programming*, pages 75 – 98, Cambridge, MA, 1994. MIT Press.

[29] P. Nordin, W. Banzhaf, and F.D. Francone. Introns in nature and in simulated structure evolution. In D. Lundh, B. Olsson, and A. Narayanan, editors, *Proceedings of Biocomputing and Emergent Computation (BCEC97), Skovde, Sweden, September 1-2, 1997*, pages 22 – 35, Singapore, 1997. World Scientific.

[30] W Langdon and W Banzhaf. Repeated patterns in genetic programming. *Natural Computing*, 7:589 – 613, 2008.

[31] P. Nordin and W. Banzhaf. Complexity compression and Evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, San Francisco, CA, USA, 1995. Morgan Kaufmann.

[32] T Soule and JA Foster. Removal bias: A new cause of code growth in tree-based evolutionary programming. In *IEEE Intl. Conference on Evolutionary Computation*, pages 781 – 786, New York, 1998. IEEE Press.

[33] W Langdon. Fitness causes bloat. Technical report, CSRP-97-22. University of Birmingham, United Kingdom, 1997.

[34] A Wagner. *Robustness and Evolvability in Living Systems*. Princeton University Press, Princeton, NJ, 2005.

[35] M Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, Cambridge, UK, 1983.

[36] C Forst, C Reidys, and J Weber. Evolutionary dynamics and optimization. In F Moran, A Moreno, JJ Merelo, and P Chacon, editors, *Advances in Artificial Life*, pages 128 – 147, Berlin, 1995. Springer.

[37] T Hu and W Banzhaf. Evolvability and speed of evolutionary algorithms in light of recent developments in biology. *Journal of Artificial Evolution and Applications*, 2010:568375 (available from http://www.hindawi.com/archive/2010/568375/), 2010.

[38] T Hu, J Payne, J Moore, and W Banzhaf. Robustness, evolvability, and accessibility in linear genetic programming. In S. Silva, JA. Foster, and et al, editors, *Proc EuroGP 2011*, pages 13 – 24, Berlin, 2011. Springer.

[39] T Hu, JL Payne, W Banzhaf, and JH Moore. Evolutionary dynamics on multiple scales. *Genetic Programming and Evolvable Machines*, 13:305 – 337, 2012.

[40] D Goldberg. *Genetic Algorithms in Search, Optimization and Machine*

Learning. Addison Wesley, Reading, MA, 1989.

[41] M Vose. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, MA, 1999.

[42] R Poli, L Vanneschi, WB Langdon, and N Freitag. Theoretical results in genetic programming: The next ten years. *Genetic Programming and Evolvable Machines*, 11:285 – 320, 2010.

[43] W Langdon and R Poli. Boolean functions fitness spaces. In R Poli, P Nordin, W Langdon, and T Fogarty, editors, *Proceedings EuroGP'99*, pages 1 – 14, Berlin, 1999. Springer.

[44] M O'Neill, L Vanneschi, S Gustafson, and W Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11:339 – 363, 2010.

[45] MJ West-Eberhard. *Developmental Plasticity and Evolution*. Oxford University Press, New York, 2003.

[46] S Ben-Tabou de Leon and EH Davidson. Gene regulation: Gene control network in development. *Annual Reviews in Biophysics and Biomolecular Structure*, 36:191 – 212, 2007.

[47] L Spector and K Stoffel. Ontogenetic programming. In J Koza, DE Goldberg, DB Fogel, and R Riolo, editors, *Proc Genetic Programming 1996*, pages 394 – 399, Cambridge, MA, 1996. MIT Press.

[48] NW Daw, NE Berman, and M Ariel. Interaction of critical periods in the visual cortex of kittens. *Science*, 199:565 – 567, 1978.

[49] S Harding, J Miller, and W Banzhaf. Evolution, development and learn-

ing using self-modifying cartesian genetic programming. In F. Rothlauf and et al, editors, *Proc. of the 11th Intl. conference on Genetic and Evolutionary Computation.*, pages 699 – 706, New York, 2009. ACM Press.

[50] J Koza, FH Bennett, D Andre, and MA Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, San Francisco, 1999.

[51] A Kattan and R Poli. Evolution of human-competitive lossless compression algorithms with gp-zip2. *Genetic Programming and Evolvable Machines*, 12:1 – 30, 2011.

[52] C Ryan. *Automatic Re-engineering of Software Using Genetic Programming.* Kluwer Academic, Boston, MA, 2000.

[53] S Forrest, TV Nguyen, W Weimer, and C Le Goues. A genetic programming approach to automated software repair. In F. Rothlauf and et al, editors, *Proc. of the 11th Intl. conference on Genetic and Evolutionary Computation*, pages 947 – 954, New York, 2009. ACM Press.

[54] B. Bhanu and Y Lin. Object detection in multi-modal images using genetic programming. *Applied Soft Computing*, 4:175 – 201, 2004.

[55] M Zhang, U Bhowan, and N Bunna. Genetic programming for object detection: A two-phase approach with an improved fitness function. *Electronic Letters on Computer Vision and Image Analysis*, 6:27 – 43, 2007.

[56] R. Poli. Genetic programming for feature detection and image segmentation. In T Fogarty, editor, *Evolutionary Computation*, pages 110 – 125, Berlin, 1996. Springer.

[57] S. Harding and W Banzhaf. Genetic programming on gpus for image processing. *International Journal of High-Performance Systems Architecture*, 1:231 – 240, 2008.

[58] W. Fu, M. Johnston, and M Zhang. Genetic programming for edge detection: A global approach. In *Proceedings of CEC 2011*, pages 254 – 261, New York, 2011. IEEE Press.

[59] G. Olague and L. Trujillo. Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. *Image and Vision Computing*, 29:484 – 498, 2011.

[60] A. Song and V. Ciesielski. Texture segmentation by genetic programming. *Evolutionary Computation*, 16:461 – 481, 2008.

[61] M Harman and A. Mansouri. Search based software engineering: Introduction to the special issue of the. *IEEE Transactions on Software Engineering*, 36:737 –741, 2010.

[62] PostgreSQL. Manual, version 9.1. http://www.postgresql.org/docs/9.1/static/geqo-pg-intro.html, Accessed 25.6.2012.

[63] H2. H2 database - features. http://www.h2database.com/html/features.html, Accessed 25.6.2012.

[64] R. Stadelhofer, W. Banzhaf, and D. Suter. Evolving blackbox quantum algorithms using genetic programming. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 22:285–297, 2008.

[65] F. Archetti, I. Giordani, and L. Vanneschi. Genetic programming for QSAR investigation of docking energy. *Applied Soft Computing*, 10:170–

182, 2010.

[66] N. Naderi, P. Roshani, M.Z. Samani, and M.A. Tutunchian. Application of genetic programming for estimation of soil compaction parameters. *Applied Mechanics and Materials*, 147:70–74, 2012.

[67] M. Ali Ghorbani, R. Khatibi, A. Aytek, O. Makarynskyy, and J. Shiri. Sea water level forecasting using genetic programming and comparing the performance with artificial neural networks. *Computers & Geosciences*, 36:620–627, 2010.

[68] P. Charbonneau. Genetic algorithms in astronomy and astrophysics. *Astrophysical Journal Supplement Series*, 101:309 – 334, 1995.

[69] J. Li, X. Yao, C. Frayn, H. Khosroshahi, and S. Raychaudhury. An evolutionary approach to modeling radial brightness distributions in elliptical galaxies. In X. Yao, E. Burke, J.A. Lozano, J. Smith, J.J. Merelo-Guervs, J.A. Bullinaria, J. Rowe, P. Tino, A. Kabn, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature-PPSN VIII*, pages 591–601. Springer, 2004.

[70] W.P. Worzel, J. Yu, A.A. Almal, and A.M. Chinnaiyan. Applications of genetic programming in cancer research. *International Journal of Biochemistry & Cell Biology*, 41:405–413, 2009.

[71] S.M. Winkler, M. Affenzeller, and S. Wagner. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines*, 10:111–140, 2009.

[72] S. Silva and L. Vanneschi. State-of-the-art genetic programming for predicting human oral bioavailability of drugs. *Advances in Bioinformatics*, 74:165–173, 2010.

[73] J Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11:251 – 284, 2010.

[74] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5:17–26, 2001.

[75] P.G. Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40:121–144, 2010.

[76] D. Dasgupta and Z Michalewicz, editors. *Evolutionary Algorithms in Engineering Applications*. Springer, 1997.

[77] L. Gusel and M. Brezocnik. Application of genetic programming for modelling of material characteristics. *Expert Systems with Applications*, 38:15014 – 15019, 2011.

[78] J Lohn, GS Hornby, and DS Linden. An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission. In UM O'Reilly, RL Riolo, G Yu, and W Worzel, editors, *Genetic Programming Theory and Practice II*, pages 301 – 315. Kluwer Academic, Boston, MA, 2004.

[79] JU Dolinsky, ID Jenksinson, and GJ Cloquhoun. Application of genetic programming to the calibration of industrial robots. *Computers in In-*

dustry, 58:255 – 264, 2007.

[80] H. Iba and T. Sasaki. Using genetic programming to predict financial data. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC-1999)*, pages 244 – 251, New York, 1999. IEEE Press.

[81] T.E. McKee and T. Lensberg. Genetic programming and rough sets: A hybrid approach to bankruptcy classification. *European Journal of Operational Research*, 138:436–451, 2002.

[82] S.H. Chen. *Genetic Algorithms and Genetic Programming in Computational Finance*. Kluwer Academic, 2002.

[83] J. Wang. Trading and Hedging in S&P 500 Spot and Futures Markets using Genetic Programming. *Journal of Futures Markets*, 20:911–942, 2000.

[84] BK Wong and TA Bodnovich. A Bibliography of Genetic Algorithm Business Application Research: 1988 - June 1996. *Expert Systems*, 15:75 – 82, 1998.

[85] M. Álvarez-Díaz and G. Caballero Miguez. The quality of institutions: A genetic programming approach. *Economic Modelling*, 25:161–169, 2008.

[86] M. Alvarez-Diaz, J. Mateu-Sbert, and J. Rossello-Nadal. Forecasting tourist arrivals to balearic islands using genetic programming. *International Journal of Computational Economics and Econometrics*, 1:64–75, 2009.

[87] L. Gritz and J. Hahn. Genetic programming for articulated figure motion. *Journal of Visualization and Computer Animation*, 6:129 – 142, 1995.

[88] K.T. Sun, Y.C. Lin, C.Y. Wu, and Y.M. Huang. An application of the genetic programming technique to strategy development. *Expert Systems with Applications*, 36:5157–5161, 2009.

[89] Y. Azaria and M. Sipper. GP-gammon: Genetically programming backgammon players. *Genetic Programming and Evolvable Machines*, 6:283–300, 2005.

[90] M Sipper and M Giacobini. Introduction to special section on evolutionary computation in games. *Genetic Programming and Evolvable Machines*, 9:279 – 280, 2008.

[91] C.G. Johnson and J.J.R. Cardalda. Genetic algorithms in visual art and music. *Leonardo*, 35:175–184, 2002.

[92] B. Johanson and R. Poli. GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters. In *Genetic Programming: Proceedings 3rd annual conference*, page 181, San Francisco, 1998. Morgan Kaufmann.

[93] W Banzhaf, S Harding, WB Langdon, and G Wilson. Accelerating Genetic Programming through Graphics Processing Units. In R Riolo, T Soule, and B Worzel, editors, *Genetic Programming Theory and Practice VI*, 1 - 23. Springer, New York, 2009.

[94] D. Robilliard, V. Marion, and C. Fonlupt. High performance genetic programming on gpu. In *Proceedings of the 2009 workshop on Bio-inspired algorithms for distributed systems*, pages 85–94, New York, 2009. ACM Press.

[95] W. Langdon. A many threaded CUDA interpreter for genetic programming. *Genetic Programming and Evolvable Machines*, 11:146–158, 2010.

[96] W.B. Langdon and A.P. Harrison. GP on SPMD parallel graphics hardware for mega bioinformatics data mining. *Soft Computing*, 12:1169–1183, 2008.

[97] J Koza, M. Keane, MJ Streeter, W Mydlowec, J Yu, G Lanza, and D. Fletcher. *Genetic Programming IV:Routine Human-competitive Machine Intelligence*. Kluwer Academic, Norvell, MA, 2003.

[98] L Spector, DM Clark, I Lindsay, B. Barr, and J Klein. Genetic Programming for Finite Algebras. In M Keijzer and et al, editors, *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1291 – 1298, New York, 2008. ACM Press.

[99] M. Keane, J Koza, and M. Streeter. *Improved General-Purpose Controllers*. US Patent Nr. 6,847,851, Jan 25 2005.

Bio

Wolfgang Banzhaf is currently University Research Professor at Memorial University of Newfoundland. From 2003 to 2009 he served as Head of the Department of Computer Science at Memorial. From 1993 to 2003 he was an Associate Professor for Applied Computer Science at Technical University of Dortmund, Germany. He also worked in industry, as a researcher with the Mitsubishi Electric Corporation in Japan and the US. He holds a PhD in

Physics from the University of Karlruhe in Germany. His research interests are in the field of bio-inspired computing, notably evolutionary computation and complex adaptive systems.