Foundations of gnetic Programming (2002)

William B Langdon
Ricardo Poli

1. Introduction.

Related approaches include neural networks, classical Artificial Intelligence and Genetic Algorithms which is encompassed by Genetic Programming. Sexual reproduction allows successful characteristics to come together in one organism and ultimately spread within a population. Copying genes is noisy and there is no explicit goal.

Problem Solving as Search. The search space is often huge, if not infinite, but often neither continuous or differentiable which invalidates many classical search techniques. GP search can be interpreted as beginning with random individuals and progressively producing fitter offspring at a point between each parent.

Microscopic Dynamical System Models. Markov chain analysis of genetic chain populations over time. Markov models for GP have been proposed by Riccardo Poli.

Fitness Landscapes. Greedy search techniques are direct but suboptimal. GA overcomes some of these issues.

Component Analysis. Considers the surviving components that result when bit strings are subjected to a crossover procedure.

Schema Theories. A schemata groups together all the points in a search space that have certain macroscopic quantities in common. These subsets are then used to study how and why individuals move from one subspace to another.

No Free Lunch Theorems. NFL theorems prove than no search algorithm is superior to all others in all search spaces. However matching the algorithm to the search space can deliver very different results.

What is Genetic Programming? In artificial evolution, most of the computer resources are devoted to deciding which individuals will have children. In GP the individuals are computer programs tested for fitness against a problem set with desired output specified by the user. In GP the programs are usually written as syntax trees or as corresponding expressions in syntax trees. New programs are created by mutation (random function with same arguments) or crossover (insert whole tree branched from one parent or another).

Tree Based Genetic Programming. Example at ftp://ftp.cs.bham/ac/uk/pub/authors/W.B.Langdon/gp-code/simple

Modular and Multiple Tree Genetic Programming. Other extensions to Koza'a basic approach include techniques to build and evolve functions (including parameters), to identify new valuable primitives or even to evolve a program which build programs.

Linear genetic Programming. Rather than being tree based LGP is typically stack based, register based or machine code (typically 10-20 times faster).

Graphical Genetic Programming. Parallel Distributed Genetic Programming (PDGP) represents programs as graphs with the edges as directed data flow between processing nodes. Shown to produce better performance in a number of benchmark problems. Parallel Architecture Discovery and Orchestration PADO defines a similar but looser structure.

Outline of the Book.

2. Fitness Landscapes.

A visualization metaphor.

Exhaustive Search.

Hill Climbing.

Fitness Landscapes as Models of Problem Difficulty. The success of a particular strategy is dependent on the topography of the fitness landscape.

An Example GP Fitness Landscape. It is difficult to use the fitness landscape metaphor because the genetic process changes the structure of the solution.

Other Search Strategies. Vary jump size, simulated annealing, random jump from peak, A*.

Difficulties with the Fitness Landscape Metaphor. One of the goals of genetic algorithms is to discover and exploit regularities. Schema analysis looks at the fitness of patterns and how well a genetic algorithm (such as GP) is exploiting them. There are problems visualizing many dimensions and binary values. Often there is no natural ordering of neighboring points. Some search spaces are not symmetric in the sense that some points can only be reached along a particular path, or that some paths can only be traversed in one direction.

Effect of Representation Changes. For example: using a binary representation for integers will radically change the nature of the landscape.

Summary. The fitness Landscape metaphor is appealing but complex search techniques render the metaphor largely redundant.

3. Program Component Schema Theories.

John Holland's mid-70's work on schemata is well know but has received recent criticism. Markov models for GA convergence are accurate but intractable and impractical. Extending Holland's schemata theorem for trees is an obvious approach but is difficult for GP where the trees are not rooted in that copies of branches can exist at multiple locations at different levels in the tree.

Prices Selection and Covariance Theorem. From population genetics relates the change in frequency in a gene in a population from one generation to the next, to the covariance between the gene's frequency in the original population and the number of offspring produced by individuals in that population. $\Delta Q = cov(z,q)/\check{z}$. For Prices theorem to hold for GP the population must be large (to avoid drift issues) and the selection of crossover and mutation points must be random and not related to fitness.

Genetic Algorithm Schemata. In GA operating on binary strings, a schema is a string of symbols {0,1,#} where # is interpreted as the “don't care” symbol. The number of non # symbols is called the o*rder* of a schema and the distance between the furthest two # symbols is called the *defining length* of the schema.

From GA Schemata to GP Schemata. Crossover mixes and concatenates the components of low order schemata to form higher order ones. A schema #10#1 could be represented with positional information as [(10,2),(1,5)]. Counting the number of strings *matching* a given schema and counting the number of schema components or *instantiation*s in the population give the same result. However, if positional information is omitted then the matches and instantiations become different. For example; the component c1 =10 is present twice in the string 10101 and the components of the schema [10,01] are jointly present four times in the string 10101. Another example; in the population {11001,00011,00001,11011} we have m([11],t) = 3, i([11],t) = 4, m([0,1],t) = 4 and i([0,1],t) = 20.

Koza's Genetic Programming Schemata. Considers a schema to be made up of multiple sub expressions but with no positional information. ie There are two instantiations of H=[(- 2 x), x] in the program (+ (- 2 x) x).

Altenberg's GP Schema Theory. Was the first mathematical formulation of a schema theorem for GP. It is a probabilistic model that treats a schema as a single sub expression. During a crossover operation, it considers the probability of producing a particular program as a result of the probabilities of selecting a particular sub-expression from one parent and placing it in a particular location in the offspring.

O'Reilly's Genetic Programming Schemata. Extended Koza's work based on the ides of defining a schema as an unordered collection of subtrees and tree fragments (trees with at least one #). This scheme, like Koza's, defines the components but not the position. Oreilly argued that the intrinsic variability of the probability that a schema will be disrupted is one of the major reasons why no hypothesis can be made on the real propagation and use of building blocks in GP.

Whigham's Genetic Programming Schemata. Is based on context free grammars CFG-GP. In CFG-GP programs are the result of applying a set of rewrite rules from the user supplied grammar. The tree's internal nodes are rewrite rules and the leaf nodes are functions and terminals used in the program. Whigham defines a schema as a partial derivation tree rooted in some non-terminal node.

Summary. In these early approaches schemata were mostly interpreted as components or groups of components which could propagate within the population.

4. Pessimistic GP Schema Theories.

This chapter introduces the Authors schema which is a generalization of the corresponding GA operator where the same crossover point is selected in both parents.

Rosca's Rooted Tree Schemata. Rosca independently proposed rooted tree-schemata which reintroduces positional information.

Fixed-Size-and-Shape Schemata in GP. For schemata to be useful in explaining how GP searches, their definition must make the effects of selection, crossover and mutation comprehensible and relatively easy to communicate. In standard fixed-length linear GA's, it is possible to contain all of the schemata contained in a string by simply replacing, in all possible ways, its symbols with “dont care” symbols which represent a single character. The same approach works for GP's. A GP schema H represents multiple programs, all having the same shape as the tree representing H and the labels for the non-= nodes. *Order* is the number of non-= symbols. *Length* is the total number of nodes in the schema. *Defining Length* is the number of links in the minimum tree fragment all the non-= symbols. The number of programs able to be represented is huge but finite and easily calculable. This schema is lower level in that it can represent the others presented. GA schemata are usually interpreted as faces and edges on a unit hypercube, the vertices's of which are all the possible bitstrigs of prefixed length. Similarly GP trees are interpreted as Hyper-spaces and Hyperplanes.

Point Mutation and One-Point Crossover in GP. In binary GA's, one-point-crossover aligns two strings, makes a common break, swaps the parts and rejoins. In GP, one-point-crossover with programs of the same shape is similar. In programs of different shape the two parents are jointly and recursively traversed from the root until a match is located. A common break point is chosen randomly in the traversed region of matching structure and the branches are swapped. In the absence of mutation the population converges to all be identical as the upper part initially converges followed progressively by lower parts. The depth of the offspring remain between the shallowest and deepest of the parents in the initial population. Common structure in the parents is respectfully preserved in the offspring. The calculations to model the disruption of schemata are feasible.

Disruption-Survival GP Schema Theorem. The theorem provides a pessimistic lower bound on the number of copies of a schema in the next generation. In GP with one-point-crossover, the propagation of a schema H depends not only on the features of the programs sampling it but also on the features of the programs having the same structure as H. The fragility of the shape of the schema H in the population also plays an important role. There will be a high rate of disruption in a population which has just been initialized in some randomized way. After one-point crossover has been active in a population for some time (with low mutation) diversity and fragility will reduce and the one-point-crossover operator will be acting on programs with the same size and shape. In this late stage GP approaches the behavior of GA.

5. Exact GP Schema Theorems.

Criticisms of Schema Theorems. Hollands schema theorem has been widely criticized as useless because the theorems have only provided a lower bound of the expected value of the number of instances of schema H in the next generation. Focus has moved to Markov models but recent results in GA can be extended to GP to provide precise solutions with much more value.

The Role of Schema Creation. Schema Creation has been ignored in the theorems so far. Transmission probability is hard to calculate but if we did have it then ...... The chance of each child matching schema H is binomially distributed so a mean, standard deviation, signal to noise ratio, probability of exactly x instances at the next generation (inc x=0=extinction) could be calculated.

Stephens and Waelbroek's GA scema Theory. Expresses transmission probability exactly for GA with one-point crossover applied.

GP Hyperschema Theory. Can be developed using the GA techniques.

- **Theory for Programs of Fixed Size and Shape.** When the populations of programs all have the same size and shape, it is possible to express the total transmission probability of a fixed size and shape schema H, α(H,t), in the presence of one point crossover. This is done by splitting the program tree into a Lower and Upper portion at the crossover point. The result of a union between two parents will only sample the schema H if the first parent has the correct lower part and the second parent the correct upper part.
    - l(H,i) is the schema obtained by replacing all the nodes above the crossover point i with = nodes
    - u(H,i) is the schema obtained by replacing all the nodes below the crossover point i with = nodes
- **Hyper-schemata.** The hyper-schema terminal set is the GP terminal set plus = and #. The operator = is a “don't care” symbol which stands for exactly one node, while the new operator # stands for any valid subtree.
- **Microscopic Exact GP Schema Theorem.** The approach can then be extended to populations including programs of different sizes and shapes. Microscopic in the sense that it considers every program in the population.
    - L(H,i) is the hyper-schema obtained by replacing all the nodes on the path between crossover point i and the root node with = nodes, and all the subtrees connected to those nodes with # nodes.

    - U(H,i) is the hyper-schema obtained by replacing the subtree below crossover point i with a # node.
- **Macroscopic Schema Theorem with Schema Creation.** The exact microscopic theory can be transformed into an exact macroscopic theory (dealing with global properties of the population).
- **Examples**. With simple non-trivial examples because the calculations are extensive to hand work.

Exact Macroscopic Schema Theorem for GP with Standard Crossover. A further extension results in a general schema for GP with subtree swapping crossover. To proceed a variable arity hyper-schema is defined as well as a Cartesian node reference system.

- **Cartesian Node Reference Systems.** can be defined by first considering the largest possible tree that can be created with nodes of arity a_{max}. Then one can organize the nodes in the tree into layers of increasing depth and assign an index to each node in a layer (depth,index). It will also be useful to transform pairs of coordinates into integers by counting the nodes in a breadth-first order.
- **Variable Arity Hyperschema**. The variable arity (VA) hyper-schemata is a rooted tree composed of internal nodes from the set FU{=,#} and leaves from TU{=,#}, where F and T are the function and terminal sets. The operator = is a “don't care” symbol which stands for exactly one node, the terminal # stands for any valid subtree, while the function # stands for exactly one function of arity not smaller than the number of subtrees connected to it.

Summary.

6. Lessons from the GP Schema Theory.

Effective Fitness. The concept of effective fitness is an attempt to rescale fitness values so as to incorporate the creative and disruptive effects of the operators. It has been independently proposed three times in the literature.

Goldberg's Operator-Adjusted Fitness in GA's. Fitness is reduced when probability of crossover is high and the defining length is a small portion of the program and/or when the probability of mutation is high and the order of the program is high. When the optima of the fitness and the operator-adjusted fitness to not coincide then the problem is said to be *deceptive*.

Nordin and Banzhaf's Effective Fitness in GP. Proposed to explain the reasons for bloat and active code suppression in GP. It is essentially the same idea as Goldberg's.

Stevens and Waebroeck's Exact Effective Fitness in GA's. Is of similar form to Goldberg's and to Nordin and Banzhafs but with the important differences that it expresses a lower bound and that effective fitness can be greater that fitness when H are abundant and relatively fit.

Exact Effective Fitness for GP. The expression for feff(H,t) is derived from that for GP with one point crossover and it can be larger than f(H,t) when the building blocks for H are relatively fit and abundant. The idea of Nordin and Banzhaf's is that individuals for which the ratio between active code and total code is smaller would appear to have higher fitness because the active code will be disrupted less often if it is “hidden” from crossover in a proportionally much larger bloat of code. The effective fitness landscape is a dynamic changing generation after generation and evolution is forced to continue until the effective fitness landscape becomes completely flat.

Operator Biases and Linkage Disequilibrium for Shapes. Standard crossover will tend to heavily sample the space of smaller than average programs and is unable to focus its search on programs of a particular size. This means that crossover bias may counteract the tendency of selection to prefer fitter programs. This does not happen with one-point crossover, which is instead completely unbiased with respect to the program length. Standard crossover for trees is generally a local search operator and one-point crossover is slightly less local, while uniform crossover (swap nodes in the common region with uniform probability) is a global search operator.

Building Blocks in GA's and GP. The building block hypothesis is typically stated informally by saying that a genetic algorithm works by combining short, low-order schemata of above average fitness to form higher-order ones over and over again until it converges to an optimum or near optimum solution. This has been highly criticized but the criticism may not apply to one-point crossover.

Practical Ideas Inspired by Schema Theories. Knowledge of operator bias is important and may be able to be combined with different initialization approaches to achieve particular results.

Convergence, Population Sizing, GP Hardness and Deception. A recursive conditional schema theorem for GA has been derived which allows one to predict the probability of obtaining a solution to a problem in a fixed number of generations assuming that the fitness of the building blocks is known and of the population is known. This may also be possible for GP.

Summary.

7. The Genetic Programming Search Space.

Conclusions.
In the first four sections of this chapter we presented experimental data from a number of common benchmark problems, which indicate that after some problem dependent size threshold, the distribution of program functionality across the whole search space tends to a limit. (The next chapter treats this formally). Informal extensions to Automatically Defined Functions (ADFs), and programs containing side effects, particularly memory and loops or recursion were given in Section 7.5. Section 7.6 considered the relationship between tree size and depth for various types of programs, while Section 7.7 discussed some of the implications for genetic programming. In general the number of programs of a given size grows approximately exponentially with that size. Thus, the number of programs with a particular fitness score or level of performance also grows exponentially; in particular the number of solutions also grows exponentially.

Chapter 8. The GP Search Space: Theoretical Analysis

We start this chapter by formally proving results given in Chapter 7 for linear and tree based GP. The rest of the chapter gives proofs for other properties of program search spaces. The implications of these results have been previously discussed in Section 7.7. For the reader who does not wish to study the proofs in detail, the main results are summarized next.

Summary. The distribution of program functionality (and hence fitness) in conventional linear programs tends to a limit as the programs get longer (Section 8.1). Furthermore convergence in the limit is exponentially fast. The threshold size is $0(1/ \log(\lambda_2))$, where λ_2 is the second largest eigenvalue of the computer's Markov state transition matrix.

Where the function set is symmetric, each function generated by long random linear programs is equally likely, and so the random chance of finding a long solution is $2^{-|\text{test set}|}$

Section 8.2 proves that, provided certain assumptions hold, there is a limiting distribution for functionality implemented by large binary tree programs (cf. Sections 7.1—7.3).

Section 8.3 proves in large trees composed only of XOR functions that the density of order n parity solutions is $1/(2^{n-1})$, provided the tree is of the right size. Also, the fitness landscape is like a needle-in-a-haystack, so any adaptive search approach will have difficulties.

Chapter 9. Example I: The Artificial Ant.

Reducing Deception. In Section 9.3 we have shown that the performance of several techniques is not much better than the best performance obtainable when using uniform random search. We suggested that this was because the program fitness landscape is difficult for hill climbers, and that the problem contains multiple levels of deception which also makes it difficult for genetic algorithms.

Analysis of high scoring non-optimal programs suggests many reach high rewards even though they exhibit poor trail following behavior. Typically they achieve high scores by following the trail for a while and then losing it at a corner or gap. They then execute a blind search until they stumble into the trail at some later point and recommence following it (see Figure 9.17). The blind search may give them a better score than a competing program that successfully navigated the same bend or gap but later lost the trail (see Figure 9.18).

Conclusions. There have often been claims that automatic programming is hampered by the nature of program spaces. These are undoubtedly large [Koza, 1992, page 2] and, it is often claimed, badly behaved with little performance relationship between similar programs [O'Reilly, 1995, page 8]. When we test these assumptions, we find that there is more to it. First, we can quantify how big search spaces actually are. Secondly, the assumption of ruggedness appears to hold only for small programs. Large programs are often interconnected by a dense network of paths connecting programs with the same fitness. (The so-called neutral networks). However, since these do not necessarily lead to programs with a better performance (i.e. high fitness), their utility is still an open question.

It is interesting that even a problem as simple as this Ant problem should have so many different solutions. We have highlighted that many of the solutions are simple transformations of each other. However, these symmetries in the search space do not match those of traditional GP genetic operators. This has similarities with the aliasing problem sometimes seen in artificial neural networks, where multiple solutions inhibit training as they have incompatible representations.

10. Example II: The Max Problem.

In this second chapter on the application of our analysis techniques, we consider a second GP benchmark problem: the “Max Problem”. Briefly, this problem is to find a program, subject to a size or depth bound, which produces the largest value [Gathercole, 1998].~ This problem is unusual in several respects: we know in advance what the solution is, what value it returns and and exactly how many solutions there are. Also, the size or depth limit is fundamental to the problem. MAX is hard for GP because of the way deception interacts with size and depth bounds to make it difficult to evolve solutions.

[Gathercole and Ross, 1996] introduced the MAX problem to GP to highlight deficiencies in the standard GP crossover operator which result from the practical requirement to limit the size of evolved programs. They concentrated on the case where program trees are restricted to a maximum depth, but in [Gathercole, 1998] showed that similar effects occur when programs are restricted to a maximum number of nodes.

The MAX problem has known optimal solutions which are composed of regularly arranged subtrees or building blocks. Despite this, GP finds solving larger versions of the MAX problem difficult.

Here we extend [Gathercole and Ross, 1996] by considering bigger trees, different selection pressures, different initializations of the population, measuring the frequency with which the depth limit affects individual crossovers, and measuring population variety and the number of steps required to solve the MAX problem. Qualitative models of crossover and population variety are presented and compared with measurements. We give an improved explanation for the premature convergence noted by [Gathercole and Ross, 1996], and this leads to the realization that there are two separate reasons why GP finds the MAX problem hard. (1) the tendency for GP populations to converge in the first few generations to suboptimal solutions from which they can never escape. (2) convergence to suboptima from which escape can only be made by slow search similar to randomized hill climbing.

10.1 The MAX Problem. In the MAX problem, “the task is to find the program which returns the largest value for a given terminal and function set and with a depth limit, D, where the root node counts as depth 0” [Gathercole and Ross, 1996, page 291]. In this chapter we use the function set { +, x }, and there is one terminal 0.5. For a tree to produce the largest value, the + nodes must be used with 0.5 to assemble subtrees with the value of 2.0. These can then be connected via either + or x to give 4.0. Finally, the rest of the tree needs to be composed only of x nodes to yield the maximum value of 42D ~ Every component of the evolved programs contributes to their fitness, and so no part can escape the effect of selection. That is, there can be no introns.

10.7 Conclusions. The analysis shows on the larger MAX problems that GP has two serious problems: 1. GP populations have a significant risk of losing vital components of the solutions at the very beginning of the run, and these components cannot be recovered later in the run. We have used Price’s Theorem to analyse why this happens and which runs will be affected. (Similar effects are reported in [Langdon, 1998c]). 2. Where solution is possible, the later stages of GP runs are effectively performing randomised hill climbing, and so solution time grows exponentially with depth of the solution.

We have extended the analysis of the difficulties that crossover experiences presented in [Gathercole and Ross, 1996] to include a quantitative model of the later evolution of MAX problem populations. Comparison with experimental results indicates that the model gives a reasonable approximation of the rate of improvement.

While [Gathercole and Ross, 1996] suggest that MAX problem populations converge, measurements indicate that after many generations up to two-thirds of the population are unique, depending upon the maximum depth and selection pressure. A model of this based on the interaction between crossover and the depth restriction has been presented. We shall return to convergence in GP in Chapter 11. The interaction of depth (and size) limits has been discounted in GP; the MAX problem clearly shows they can have an important impact (we return to this in Section 11.4). Additionally, a model of the number of duplicate individuals in the initial populations has been presented which highlights a potential problem with the standard technique for generating the initial random population.

In Section 10.6 we used the MAX problem to demonstrate the applicability of Price’s covariance and selection theorem of gene frequencies to GP populations, but noted that GP’s depth restriction influences crossover and the consequent implications on the changes in gene frequencies. Gene covariance was analyzed to help explain why GP populations get locked into suboptimal solutions in the first few generations.

Chapter 11: GP Convergence and Bloat.

11.7 Conclusions. We have advanced the idea that bloat is a form of convergence in genetic programming. With populations of genotypes evolving along neutral networks towards the largest accessible part of the search space in an entropy-driven process like a random walk. In the case of binary function sets, we have mapped out the space and shown that populations evolve roughly along a near parabolic ridge at a subquadratic rate.

The average growth in program depth when using standard subtree crossover with standard random initial trees is almost linear. The rate is about one level per generation, but varies between problems. When combined with the known distribution of number of programs, this yields a prediction of subquadratic, rising to quadratic, growth in program size. There is considerable variation between runs, but on the average we observe subquadratic growth in size. In a continuous domain problem this rises apparently towards a quadratic power law limit, i.e.

Discrete mean length~O(generations2°)

Continuous lim0 ~ mean length=O (generations2 .0)

However in large discrete problems we observe a new type of genetic programming specific convergence: the whole population has the same fitness, even though its variety is 100%. This reduces the selection pressure on the population which, we suggest, is the reason why the growth remains subquadratic.

Most GP systems store each program separately and memory usage grows linearly with program size; i.e. 0 (gellerations’ 2_2). Run time is typically dominated by program execution time, which is proportional to program length [Langdon, 1998c, Appendix D.8], therefore run time is 0(generations223).

In other systems the whole population is stored in a directed acyclic graph (DAG) [Handley, 1994]. New links are created at a constant rate. However, memory usage may be less than 0(generations), since in every generation programs are deleted. In the absence of side effects and with a fixed fitness function, it may be possible to avoid re-evaluating unchanged code by caching intermediate values. That is, only code from the crossover point to the root would be executed. Then run time should be proportional to the average height of trees. So run time 0(generations2).

Note we refer to standard subtree crossover; other genetic operators and/or representations have different bloat characteristics. For example, [Nordin and Banzhaf, 1995] suggest that program size increases exponentially with generations in their linear machine code representation and crossover operator. While new mutation [Langdon, 1998b] and crossover operators [Langdon, 2000] can reduce bloat in trees.

GP populations using standard subtree crossover (and no parsimony techniques) quickly reach the bounds on size or depth that are commonly used. When this will happen can be readily estimated. We suggest such bounds may have unanticipated (but problem-dependent) benefits.

To allow big programs (1,000,000 nodes) to evolve, we were restricted to simple problems which can be solved by small trees. However, our results do raise the question of how effective subtree crossover will be on complex discrete problems whose solutions are big programs. It may also be the case that subtree crossover will cease to be effective (i.e. explorative or disruptive) if the program is structured as big trees. GP may need to limit tree size (perhaps by evolving programs composed of many smaller trees) and/or alternative genetic operators may be required.

In Section 11.6 we outlined a few of the highly successful antibloat techniques and some of their potential disadvantages. So, while it might be claimed that bloat has been solved, it remains an interesting theoretical problem due to its widespread occurrence (many other phenomena are problem specific). Also, investigating bloat has lead to insights into GP dynamics and better understanding of GP biases. Which has lead to new operators with better biases.

This concluding chapter describes a conceptually simple and usable genetic programming theory, has tested it to new extremes in GP and shown, at least in part, that it works. In the case where its predictions were less than 100% correct we see an example of new GP behavior: extreme phenotypic convergence.

Further Reading

- Poli et al 2001 see 1.1.1 Markov chain models for GP
- Nordin 1997. register based and machine code GP.
- Poli 1997 & 1999a. Parallel Distributed Genetic Programming (PDGP)
-