A New Schema Theory for Genetic Programming with One-point Crossover and Point Mutation

Riccardo Poli and W.B. Langdon
School of Computer Science, The University of Birmingham
Birmingham B15 2TT, UK
{R.Poli,W.B.Langdon}@cs.bham.ac.uk
Phone: +44-121-414-3739/4791

ABSTRACT

In this paper we first review the main results obtained in the theory of schemata in Genetic Programming (GP) emphasising their strengths and weaknesses. Then we propose a new, simpler definition of the concept of schema for GP which is quite close to the original concept of schema in genetic algorithms (GAs). Along with a new form of crossover, one-point crossover, and point mutation this concept of schema has been used to derive an improved schema theorem for GP which describes the propagation of schemata from one generation to the next. In the paper we discuss this result and show that our schema theorem is the natural counterpart for GP of the schema theorem for GAs, to which it asymptotically converges.

1 Introduction

Genetic Programming (GP) has been applied successfully to a large number of difficult problems like automatic design, pattern recognition, robotic control, synthesis on neural architectures, symbolic regression, music and picture generation [Koza, 1992, Koza, 1994, K. E. Kinnear, Jr., 1994, Koza et al., 1996, Angeline and Kinnear, Jr., 1996]. However a relatively small number of theoretical results are available to try and explain why it works and how (see [Langdon, 1996, pages 517–519] for a list of references).

Since John Holland's seminal work in the mid seventies and his well known schema theorem (see [Holland, 1992] and [Goldberg, 1989]), schemata (similarity templates representing entire groups of genes) are often used to explain why GAs work (although the usefulness of them has been recently criticised [Altenberg, 1995]). So the obvious most natural way of creating a theory for GP has been to define a concept of schema for parse trees and to extend Holland's schema theorem to GP to see how schemata would propagate generation after generation under the effects of selection, crossover and mutation.

One of the difficulties in obtaining theoretical results using the idea of schema is that the definition of schema for GP is much less straightforward than for GAs and alternative definitions have been proposed in the literature [Koza, 1992, O'Reilly and Oppacher, 1995, O'Reilly, 1995, Whigham, 1995, Whigham, 1996a]. These define schemata as composed of one or multiple fragments of a tree. Therefore, a schema can be present multiple times within the same program. This, together with the variability of the size and shape of the programs matching a same schema, leads to considerable complications in the computation of schema-disruption probabilities. Nonetheless two of these definitions have been used in schema theorems for GP, which we will review in Section 2.

In this paper we propose a new simpler definition of schema for GP which is much closer to the original concept of schema in GAs. This concept of schema has suggested us to define and use a simpler form of crossover for GP in which *the same* crossover point is selected in both parents. We call this new operator one-point crossover because of its similarity with the corresponding GA operator. As described in Section 3, we have used these tools to derive a simple and natural schema theorem, in which the probability of disruption for a schema can be estimated very easily.

An analysis of the theoretical results described in the paper is reported in Section 4 while Section 5 includes some final remarks and indications of future work.

2 Previous Work on Schemata for GP

In the context of binary representations, a schema (or similarity template) is a string of symbols taken from the alphabet

{0,1,#}. The character # is a "don't care" symbol, so that a schema can represent several bit strings. For example the schema #10#1 represents four strings: 01001, 01011, 11001 and 11011. The number of non-# symbols is called the *order* $\mathcal{O}$ of a schema. The distance between the furthest two non-# symbols is called the *defining length* $\mathcal{L}$ of a schema. Holland obtained a result (the schema theorem) which predicts how the number of instances of a schema in a population varies from one generation to the next [Holland, 1992]. The theorem is as follows:

$$m(H,t+1) \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)}(1-p_m)^{\mathcal{O}(H)} \times \left[1-p_c\frac{\mathcal{L}(H)}{N-1}\left(1-\frac{m(H,t)f(H,t)}{M\,\bar{f}(t)}\right)\right] \quad (1)$$

where $m(H,t)$ is the number of strings matching the schema H at generation t, $f(H,t)$ is the mean fitness of the strings matching H, p_c is the probability of crossover, p_m is the probability of mutation per bit, $\bar{f}(t)$ is the mean fitness of the strings in the population, M is the number of strings in the population, N is the number of bits in the strings, and $m(H,t+1)$ is the expected value of the number of strings matching the schema H at generation $t+1$.[1]

Given the popularity of this result, Koza [Koza, 1992, pages 116–119] made a first attempt to explain why GP works producing an informal argument showing that Holland's schema theorem would work for GP as well. The argument was based on the idea of defining a schema as the subspace of all trees which contain, as subtrees, a predefined set of *complete* subtrees. So, according to Koza's definition, a schema H could be represented as a set of S-expressions, e.g. H={`(+ 1 x), (* x y)`} which represents all the programs including at least one occurrence of the expression `(+ 1 x)` and at least one occurrence of `(* x y)`. Koza did not produce a definition of order or defining length for GP schemata.

Koza's work on schemata was later formalised and refined by O'Reilly [O'Reilly and Oppacher, 1995, O'Reilly, 1995] who derived a schema theorem for GP in the presence of fitness-proportionate selection and crossover. The theorem was based on the idea of defining a schema as an unordered collection (a multiset) of subtrees and tree fragments. Tree fragments are trees with at least one leaf that is a "don't care" symbol ('#') which can be matched by any subtree (including subtrees with only one node). For example the schema H={`(+ # x), (* x y), (* x y)`} represents all the programs including at least one occurrence of the tree fragment `(+ # x)` and at least *two* occurrences of `(* x y)`.[2] The tree fragment `(+ # x)` is present in all programs in which an expression is added to `x`.

[1] This is a slightly different version of Holland's original theorem, which applies when crossover is performed taking both parents from the mating pool [Goldberg, 1989, Whitley, 1993].

[2] We use here the standard notation for multisets, which is slightly different from the one used in O'Reilly's work.

O'Reilly's definition of schema allowed her to define the concept of order and defining length for GP schemata. In her definition the order of a schema is the number of non-# nodes in the expressions or fragments contained in the schema; the defining length is the number of links included in the expressions and tree fragments in the schema plus the links which connect them together. Unfortunately, the definition of defining length is complicated by the fact that the components of a schema can be embedded in different ways in different programs. Therefore, the defining length of a schema is not constant but varies with the programs sampling it. As the total number of links in each tree is variable, this implies that the probability of disruption $P_d(H,h,t)$ of a schema H due to crossover depends on the shape and size of the tree h matching the schema. The schema theorem derived by O'Reilly overcame this problem by considering the maximum of such a probability, $P_d(H,t) = max_{h\in \mathrm{Pop(t)}} P_d(H,h,t)$ which, in some cases, can lead to severely underestimating the number of occurrences of the given schema in the next generation. O'Reilly discussed the usefulness of this result and argued that the intrinsic variability of $P_d(H,t)$ from generation to generation is one of the major reasons why no hypothesis can be made on the real propagation and use of building blocks (short, low-order relatively fit schemata) in GP. O'Reilly's schema theorem did not include the effects of mutation.

In the framework of his GP system based on context free grammars (CFG-GP) Peter Whigham produced a very general concept of schema for context-free grammars and the related schema theorem [Whigham, 1995, Whigham, 1996b, Whigham, 1996a]. In CFG-GP programs are the result of applying a set of rewrite rules taken from a pre-defined grammar to a starting symbol S. The process of creation of a program can be represented with a derivation tree whose internal nodes are rewrite rules and whose terminals are the functions and terminals used in the program. In CFG-GP the individuals in the population are derivation trees and the search proceeds using crossover and mutation operators specialised so as to always produce valid derivation trees.

Whigham defines a schema as a partial derivation tree rooted in some non-terminal node, i.e. as a collection of rewrite rules organised into a single derivation tree. Given that the terminals of a schema can be both terminal and non-terminal symbols of a grammar, a schema can actually represent all the programs that can be obtained by completing the schema (i.e. by adding other rules to its leaves until only terminal symbols are present) and by using it as a component for other derivation trees. This definition of schema leads to a very simple equation for the probability of disruption of schemata under crossover and mutation. Unfortunately, like in O'Reilly's case, these probabilities vary with the size of the complete derivation tree (i.e. of the program) containing the schema. To further complicate things a schema can occur multiple times in the derivation tree of a program.

Whigham overcame these problems and produced a schema theorem for CFG-GP by considering the average dis-

ruption probability and the average fitness of each schema. By changing the grammar used in CFG-GP this theorem can be shown to be applicable both to GAs with fixed length binary strings and to standard GP, of which CFG-GP is a generalisation (see the GP grammar given in [Whigham, 1996a, page 130]).

The GP schema theorem obtained by Whigham is different from the one obtained by O'Reilly as the concept of schema used by the two authors is different. Whigham's schemata represent derivation-tree fragments which always represent single subexpressions, while O'Reilly's schemata can represent multiple subexpressions.

It is interesting to note that although both definitions are very general they seem to be unable to represent simple similarity templates like `(# x y)` which represents all the programs in which `x` and `y` are used as arguments for the same function. According to Whigham's definition such programs could not be represented by a single schema, while according to O'Reilly's definition they could only be represented, as a whole, by H=`{x,y}` which also represents all the programs in which `x` and `y` are used in any possible way.

3 GP Schema Theory

In this section we will describe our new definition of schema for GP, a new simple form of crossover, one-point crossover, and a new schema theorem for GP. Our schema theorem describes how the number of programs contained in a schema varies from one generation to the next, as a result of fitness proportionate selection, one-point crossover and point mutation.

3.1 Definition of Schema

Let $\mathcal{F}$ and $\mathcal{T}$ be the function set and the terminal set used in a GP run, respectively. We define a *schema* as a tree composed of functions and terminals from the function set $\mathcal{F} \cup \{=\}$ and terminal set $\mathcal{T} \cup \{=\}$, respectively. The symbol $=$ is a "don't care" symbol which stands for a single terminal or function (not an entire subtree, supertree or derivation tree like in O'Reilly's and Whigham's definitions).

Very much in line with the original definition of schema for GAs, a schema H can represent several programs, all having the same shape as the tree representing H and the same labels for the non-$=$ nodes. For example, if $\mathcal{F}$=`{+, -}` and $\mathcal{T}$=`{x, y}` the schema H=`(+ (- = y) =)` would represent the four programs `(+ (- x y) x)`, `(+ (- x y) y)`, `(+ (- y y) x)` and `(+ (- y y) y)`.

The number of non-$=$ symbols is called the *order* $\mathcal{O}(H)$ of a schema H, while the total number of nodes in the schema is called the *length* $N(H)$ of the schema. The length of a schema H is also equal to the number of nodes in the programs matching H.

Schemata of low order and large length can represent a huge number of programs. For example, if only two symbols are present in the function and terminal sets, a schema of order $\mathcal{O} = 5$ and length $N = 30$ represents $2^{N-\mathcal{O}} = 33,554,432$ different programs. Likewise, given that a program of length N includes 2^N different schemata, a modest-size population of programs can actually sample a huge number of schemata.

The number of links in the minimum subtree including all the non-$=$ symbols within a schema H is called the *defining length* $\mathcal{L}(H)$ of the schema. The defining length $\mathcal{L}(H)$ can be computed with a simple recursive procedure, not dissimilar from the one necessary to compute $N(H)$. Figure 1 shows some of the 32 schemata matching the program `(+ (- 2 y) x)` along with their order and defining length.

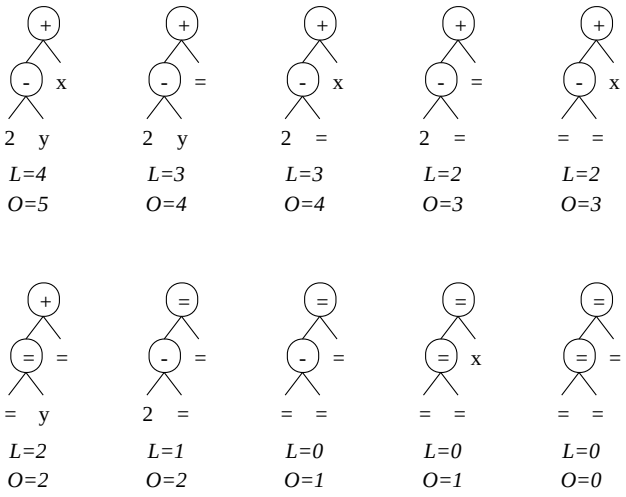

Figure 1: Some schemata sampling the program `(+ (- 2 y) x)`.

Our definition of schemata is in some sense lower-level than those adopted by O'Reilly and Whigham as a smaller number of trees can be represented by schemata with the same number of "don't care" symbols. This means that it is possible to represent one of their schemata by using a collection of ours. For example, assuming that the maximum allowed depth for trees is 2 and that all functions have arity 2, O'Reilly's schema H=`{(+ # #)}` can be represented by our schemata `(+ = =)`, `(= = (+ = =))`, `(= (+ = =) =)`, `(= (+ = =) (= = =))`, `(= (= = =) (+ = =))`, `(+ = (= = =))`, `(+ (= = =) =)` and `(+ (= = =) (= = =))`. The converse is not true, as some of our schemata (for example the similarity template at the end of Section 2) cannot be represented using O'Reilly's and Whigham's schemata.

It is also worth noting that our definitions of order, length and, in particular, defining length of a schema are totally independent on the shape and size of the programs in the actual population, unlike O'Reilly's and Whigham's.

3.2 Point Mutation and One-Point Crossover in GP

The concept of schema just introduced resembles so much the one of GA schema that we started wondering whether it would have been possible to define operators similar to the ones used in Holland's work. The obvious analogue of bit-flip mutation is the substitution of a function in the tree with another function with the same arity, or the substitution of a terminal with another terminal: a technique that has been sometimes used in the GP literature [McKay et al., 1995]. The perhaps less obvious equivalent of one-point crossover for bit strings (in which a common crossover point is selected in both parents, and the offspring are produced by swapping the bits on the right of the crossover point) is a new crossover operator that we also call *one-point crossover*.

One-point crossover works by selecting a common crossover point in the parent programs and then swapping the corresponding subtrees like standard crossover. In order to account for the possible structural differences between the two parents, one-point crossover involves two phases. Firstly, the two parent trees are recursively traversed to identify the parts with the same topology, i.e. with the same arity in the nodes encountered traversing the trees from the root node. Recursion is stopped whenever an arity mismatch between corresponding nodes in the two trees is present. All the links traversed in this phase are stored in a vector. Secondly, a random crossover point is selected among such links with a uniform probability. Figure 2 illustrates a possible behaviour of one point crossover on the programs

```
(pdiv (pdiv (- 2 x) (+ 1 (- 1 1)))
      (- 2 (pdiv (pdiv 1 x) 1)))
```

and

```
(* (+ x (+ (pdiv 1 2) 1))
   (+ (+ y (+ x x)) (* 1 1))).
```

It should be noted that this form of crossover does not increase the depth of the offspring beyond that of the parents. This feature may be useful to avoid the typical undesirable growth of program size, which can easily lead to overfitting. Obviously this means that some care is needed when selecting the maximum tree depth to be used in the creation of the initial population.

One-point crossover resembles the strong context preserving crossover operator proposed in [D'haeseleer, 1994], which is, however, less restrictive than one-point crossover as to which links can be selected as crossover points.

3.3 Schema Theorem

3.3.1 Effect of Fitness Proportionate Selection

The effect of fitness proportionate selection on our schemata can be obtained by performing exactly the usual calculations (see for example [Holland, 1992, Goldberg, 1989]

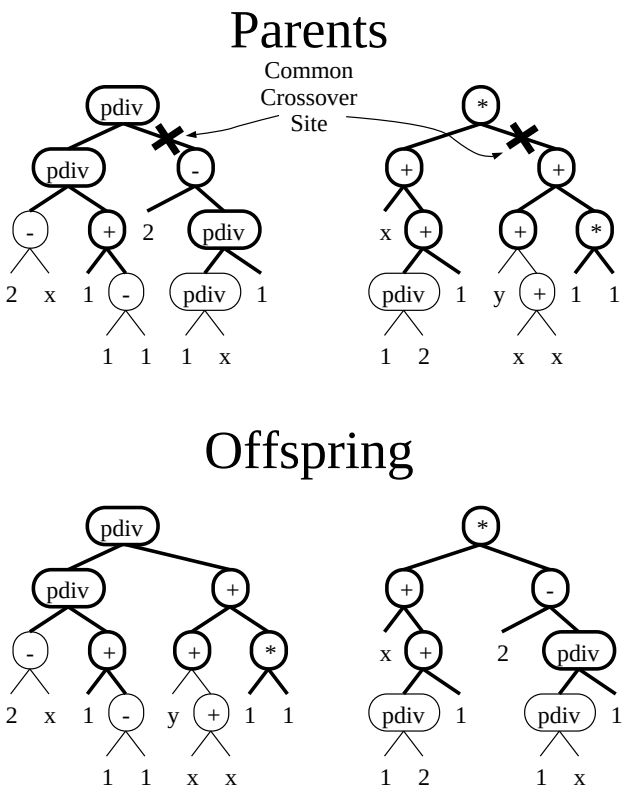

Figure 2: One point crossover (the common parts where the crossover point can be placed are drawn with thick lines).

or [Whigham, 1996a, Appendix C]). The net result is:

$$m(H, t+\frac{1}{2}) = m(H,t)\frac{f(H,t)}{\bar{f}(t)}$$

where $m(H, t+\frac{1}{2})$ is the expected value of the number of programs sampling H in the mating pool and the other symbols have the same meaning as in Equation 1.

As usual, if the population could include a large number of individuals and the ratio between schema fitness and population fitness was constant, this result would support the claim that schemata with above-average (below-average) fitness will tend to get an exponentially increasing (decreasing) number of programs sampling them. Unfortunately, this claim is often unsupported as in real-life GAs and GPs populations are always finite and the estimate of the fitness of schemata change due to the fact that the composition of the population changes from generation to generation.

3.3.2 Effect of One-point Crossover

One-point crossover can affect the propagation of schemata by disrupting some of them, ignoring others or creating some new schemata. Let us term $D_c(H)$ the event "H is disrupted when a program h sampling H is crossed over with a program $\hat{h}$" (by "disrupted" we mean that H is not sampled by the offspring produced).

There are two ways in which H can be disrupted. Firstly H can be disrupted when h is crossed over with a program $\hat{h}$ with a different structure. If we denote with $G(H)$ the zero-th order schema with the same structure of H where all the defining nodes in H have been replaced with "don't care" symbols, this can be expressed as the joint event $D_{c1}(H) = \{D_c(H), \hat{h} \notin G(H)\}$.

For example, if we consider the schema H=`(+ = =)` contained in h=`(+ x y)` and we cross h over with $\hat{h}$=`(+ z (+ 2 3))` then swapping the subtree `(+ 2 3)` with `y` would produce the program `(+ x (+ 2 3))` which does not sample $G(H)$=`(= = =)` and therefore cannot sample H.

The probability of the event D_{c1} is given by:

$$\Pr\{D_{c1}(H)\} = \Pr\{D_c(H)|\hat{h} \notin G(H)\} \Pr\{\hat{h} \notin G(H)\}$$

where

$$\Pr\{\hat{h} \notin G(H)\} = \frac{M - m(G(H), t+\frac{1}{2})}{M}.$$

The term $\Pr\{D_c(H)|\hat{h} \notin G(H)\}$, which we will denote with p_{diff} for brevity, is harder to quantify as not all crossovers between two parents h and $\hat{h}$ with different structure produce offspring which do not sample $G(H)$. We will leave this term in symbolic form to avoid using the tempting, but oversimplistic assumption that, when $\hat{h} \notin G(H)$, all crossover operations produce offspring not sampling H i.e. that $p_{\text{diff}} = 1$.

The other way in which H can be disrupted, even if h is crossed over with a program $\hat{h}$ with the same structure of H (i.e. $\hat{h} \in G(H)$), is when $\hat{h}$ does not sample H (i.e. $\hat{h} \notin H$). This can be expressed as the joint event $D_{c2}(H) = \{D_c(H), \hat{h} \in G(H)\}$

A necessary condition for $D_{c2}(H)$ is that the crossover point is between the defining nodes (the non-= nodes) of H. We will call this event $B(H)$. For example, the following table shows which schemata of h=`(+ x y)` could be disrupted in this way depending on which of the two possible crossover points in h (between `+` and `x` or between `+` and `y`) is chosen:

	Crossover Point	
Schema	`+/x`	`+/y`
`(= = =)`	Unaffected	Unaffected
`(= = y)`	Unaffected	Unaffected
`(= x =)`	Unaffected	Unaffected
`(+ = =)`	Unaffected	Unaffected
`(= x y)`	**Disrupted?**	**Disrupted?**
`(+ = y)`	Unaffected	**Disrupted?**
`(+ x =)`	**Disrupted?**	Unaffected
`(+ x y)`	**Disrupted?**	**Disrupted?**

The probability of $D_{c2}(H)$ is given by:

$$\Pr\{D_{c2}(H)\} = \Pr\{D_c(H)|\hat{h} \in G(H)\} \Pr\{\hat{h} \in G(H)\}$$

where

$$\Pr\{\hat{h} \in G(H)\} = \frac{m(G(H), t+\frac{1}{2})}{M}$$

and

$$\begin{aligned}\Pr\{D_c(H)|\hat{h} \in G(H)\} &= \Pr\{D_c(H)|\hat{h} \notin H\} \times \\ &\quad \Pr\{\hat{h} \notin H|\hat{h} \in G(H)\}\end{aligned}$$

given that $\Pr\{D_c(H)|\hat{h} \in H\} = 0$, i.e. the probability of a schema present in both parents being disrupted by crossover is zero. As

$$\Pr\{\hat{h} \notin H|\hat{h} \in G(H)\} = \frac{m(G(H), t+\frac{1}{2}) - m(H, t+\frac{1}{2})}{m(G(H), t+\frac{1}{2})}$$

and

$$\Pr\{D_c(H)|\hat{h} \notin H\} \leq \Pr\{B(H)\} = \frac{\mathcal{L}(H)}{N(H)-1},$$

we obtain:

$$\Pr\{D_{c2}(H)\} \leq \frac{\mathcal{L}(H)}{N(H)-1} \frac{m(G(H), t+\frac{1}{2}) - m(H, t+\frac{1}{2})}{M}.$$

Therefore, an upper bound for the probability of disruption for a schema H due to crossover is given by:

$$\begin{aligned}\Pr\{D_c(H)\} &\leq p_{\text{diff}} \frac{M - m(G(H), t+\frac{1}{2})}{M} \\ &+ \frac{\mathcal{L}(H)}{N(H)-1} \frac{m(G(H), t+\frac{1}{2}) - m(H, t+\frac{1}{2})}{M}.\end{aligned}$$

As crossover is applied to each program in the mating pool with a probability $p_c \leq 1$, for a proportion $1 - p_c$ of the population the number of occurrences of a schema H after crossover remains unaltered (i.e. $m(H, t+\frac{1}{2})$). For the rest of the population it is equal to $m(H, t+\frac{1}{2})$ times the probability that H is not disrupted. So,

$$\begin{aligned}m(H, t+1) &= (1-p_c) m(H, t+\frac{1}{2}) \\ &+ p_c m(H, t+\frac{1}{2}) (1 - \Pr\{D_c(H)\}) \\ &= m(H, t+\frac{1}{2})(1 - p_c \Pr\{D_c(H)\})\end{aligned}$$

3.3.3 Effect of Point Mutation

Point mutation consists of changing the label of a node into a different one. We assume that it is applied to the nodes of the programs created using selection and crossover as described in the previous sections with a probability p_m per node. A schema H will survive mutation only if *all* its $\mathcal{O}(H)$ defining nodes are unaltered. The probability that each node is not altered is $1 - p_m$. So, a schema will be disrupted by mutation with a probability:

$$\Pr\{D_m(H)\} = 1 - (1 - p_m)^{\mathcal{O}(H)}$$

This equation emphasises how high order schemata are affected by mutation more than lower order ones. If mutation,

like in GAs, is applied with a probability $p_m \ll 1$, this equation can be simplified using the first terms of its Taylor expansion:

$$\Pr\{D_m(H)\} \approx p_m \mathcal{O}(H).$$

3.3.4 Theorem

By considering all the various effects discussed above, we can write:

$$m(H,t+1) = m(H,t+\frac{1}{2})(1-\Pr\{D_m(H)\})(1-p_c \Pr\{D_c(H)\}).$$

By substituting the previous results into this expression we obtain the new **GP Schema Theorem:**

$$\begin{aligned} & m(H,t+1) \geq \\ & \quad m(H,t)\frac{f(H,t)}{\bar{f}(t)}(1-p_m)^{\mathcal{O}(H)} \times \\ & \quad \left\{1 - p_c \left[p_{\text{diff}} \left(1 - \frac{m(G(H),t) f(G(H),t)}{M\bar{f}(t)}\right) + \right.\right. \\ & \quad \left.\left. \frac{\mathcal{L}(H)}{(N(H)-1)} \frac{m(G(H),t) f(G(H),t) - m(H,t) f(H,t)}{M\bar{f}(t)} \right] \right\} \end{aligned} \tag{2}$$

which provides a lower bound for the expected number of individuals sampling a schema H at generation $t+1$ for GP with one-point crossover and point mutation.

4 Discussion

The GP schema theorem given in Equation 2 is considerably more complicated than the corresponding version for GAs in Equation 1. This is due to the fact that in GP the trees undergoing optimisation have variable size and shape. In Equation 2 this is accounted for by the presence of the terms $m(G(H),t)$ and $f(G(H),t)$, which summarise the characteristics of all the programs in the population having the same shape and size as the schema H whose propagation is being studied. Therefore in GP the propagation of a schema H does not only depend on the features of the programs sampling it but also on the features of the programs having the same structure as H. However, in GP also the probability that crossover between parents with different structure is disruptive for H, $p_{\text{diff}} = \Pr\{D_c(H) | \hat{h} \notin G(H)\}$, plays an important role.

In the following we will discuss how these terms change over time in a run and show that GP with one-point crossover tends asymptotically to behave like a GA and that, for t big enough, Equation 2 transforms into Equation 1.[3]

We should first note that crossing over an individual $\hat{h} \notin G(H)$ with $h \in H$ does not necessarily mean that H is disrupted. In fact if the common part between h and $\hat{h}$ where the crossover point can be placed includes also terminals then crossover can swap subtrees with exactly the same shape and size. This swap might or might not disrupt H depending on the actual node-composition of the subtrees swapped and on the characteristics of the corresponding subtree in H.

If we consider the typical case in which functions with different arity are present in the function set and the initialisation method is "ramped-half-and-half" or "grow" [Koza, 1992], *at the beginning of a run* it is unlikely that two trees h and $\hat{h}$ with *different* shapes undergoing crossover will swap a subtree with exactly the same shape. This is because given the diversity in the initial population only a small number of individuals will have common parts including terminals. Therefore, we can safely assume that $p_{\text{diff}} \approx 1$.

It should be noted that the term $m(G(H),t)f(G(H),t)$ represents the total fitness allocated to the programs with structure $G(H)$ (by definitions it is equal to $\sum_{h \in G(H)} f(h)$, where $f(h)$ is the fitness of individual h). Likewise $M\bar{f}(t)$ is the total fitness in the population (i.e. $\sum_{h \in Pop(t)} f(h)$). Therefore the diversity of the population at the beginning of a run implies also that $m(G(H),t)f(G(H),t) \ll M\bar{f}(t)$ and that *the probability of schema disruption is very big (i.e. close to 1) at the beginning of a run*, even disregarding the events $D_{c2}(H)$. In summary, at the beginning of a run crossover heavily counteracts the effect of selection.

Nonetheless schemata with above average fitness and short defining-length (and therefore low order) with respect to their total size will tend to survive more frequently even in this early highly disruptive phase of a run. However, *they will do so only if their shape $G(H)$ is of above average fitness (i.e. $f(G(H),t) > \bar{f}(t)$) and is shared by an above average number of programs (i.e. $m(G(H),t) > 1$).*

If the mutation rate is small after a while the population in GP with one-point crossover will start converging, like a GA, and the diversity of shapes and sizes will decrease. This in turn will make the common part between pairs of programs undergoing crossover grow and include more and more terminals. As a result, the probability of swapping subtrees with the same structure when crossing over programs with different structure will increase and p_{diff} will decrease. Losing diversity also means that a relatively small number of different program structures $G(H)$ will survive in the population and that the factor multiplying p_{diff} in Equation 2 will become less important.

Therefore, in a *second phase of a run* the probability of disruption when crossover is performed between trees with the same shape and size, $\Pr\{D_{c2}(H)\}$, will start playing an important role in determining which schemata will propagate according to their fitness and which will instead be disrupted. In particular, a schema H with above average fitness and short defining-length (and therefore low order) with respect to its total size will tend to survive better. However, also schemata with large defining length can survive *if they suffer from little competition from the schemata sampled by the programs of the same shape not containing H*. This happens when $m(G(H),t)f(G(H),t) - m(H,t)f(H,t) = \sum_{h \in G(H), h \notin H} f(h) \ll M\bar{f}(t)$.

[3] Some of the arguments presented in the rest of the section are the result of observing the actual behaviour of runs of GP with one-point crossover and also of preliminary experiments, not reported here, in which we have counted and studied the creation, disruption and propagation of the schemata present in small populations.

After many generations these effects are exacerbated: the probability of swapping subtrees with the same structure tends to 1 and p_{diff} tends to become unimportant. In this situation $m(G(H),t) = M$, $f(G(H),t) = \bar{f}(t)$ and Equation 2 becomes exactly the same as Equation 1. This is consistent with the intuition that if all the programs have the same structure then their nodes can be described with fixed-length strings and GP with one-point crossover is really nothing more than a GA.[4]

In summary, the presence of two parts in the term multiplying p_c in Equation 2 accounts for the fact that in GP with one-point crossover two competitions are going on. The first one happens at the beginning of a run when different *hyperspaces* $G(H)$ representing all programs with a given shape and size compete. In this phase the defining length $\mathcal{L}(H)$ and the number of nodes $N(H)$ of the individual *hyperplanes* H sampling the hyperspaces are nearly irrelevant: only the fitness of the programs sampling H counts. The second competition starts when the first one starts settling. In this second phase the hyperplanes within the few remaining hyperspaces start competing on the basis of their relative defining lengths as well as their fitness. In this phase GP tends to behave more and more like a standard GA.

As discussed in [O'Reilly and Oppacher, 1995], in general no schema theorem alone can support conclusions on the correctness of the *Building Block Hypothesis* (BBH), which states that GAs/GP work by hierarchically composing relatively fit, short schemata to form complete solutions. However, our definition of schema and the characterisation of $\Pr\{D_c(H)\}$ presented above seem to suggest that the BBH is much less troublesome in GP with one-point crossover than in standard GP. It is possible, however, that two kinds of building blocks are needed to explain how GP builds solutions: building blocks representing hyperspaces and building blocks representing hyperplanes within hyperspaces.

5 Conclusions

In this paper we have presented a simplified form of GP in which, thanks to constraints on the selection of crossover points, crossover transmits to the offspring many of the features common to the parents. However, rather than using the classical strategy of showing experimentally that this form of GP really works and that on a few carefully selected problems it is better than standard GP (we will do this in future papers), we start from developing a schema theory for it.

The theory is based on a new simpler definition of the concept of schema for GP which is much closer to the original concept of schema in GAs than the definitions used by other GP theorists. The simplicity of this definition along with that of one-point crossover has allowed us to derive a new schema theorem in which, for the first time, the competitions between programs with different structure and programs with the same structure have been modelled mathematically. These models have been then discussed showing that in the runs of GP with one-point crossover and point-mutation two phases are present: one in which the algorithm searches for the best shape and size for the solutions, and one in which the optimal composition in terms of functions and terminals is sought for. The algorithm moves from one phase to the next smoothly.

Our analysis shows that the new schema theorem is the natural counterpart for GP of the GA schema theorem and that the GP schema theorem asymptotically tends to it. It also suggests that the building block hypothesis, which has been strongly criticised in the context of standard GP, might in fact hold in GP with one-point crossover.

Some recent experimental results have corroborated the theory presented in this paper [Poli and Langdon, 1997]. More theoretical and experimental work will be needed to investigate the building block hypothesis.

[4] It should be noted that in less typical conditions where the "full" initialisation method is used and all functions have the same arity, then every individual in the population has the same shape and Equation 2 is the same as Equation 1 from the beginning of a run.

Acknowledgements

The authors wish to thank the members of the EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) group for useful discussions and comments. This research is partially supported by a grant under the British Council-MURST/CRUI agreement and by a contract with the Defence Research Agency in Malvern.

References

[Altenberg, 1995] Altenberg, L. (1995). The Schema Theorem and Price's Theorem. In Whitley, L. D. and Vose, M. D., editors, *Foundations of Genetic Algorithms 3*, pages 23–49, Estes Park, Colorado, USA. Morgan Kaufmann.

[Angeline and Kinnear, Jr., 1996] Angeline, P. J. and Kinnear, Jr., K. E., editors (1996). *Advances in Genetic Programming 2*. MIT Press, Cambridge, MA, USA.

[D'haeseleer, 1994] D'haeseleer, P. (1994). Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA. IEEE Press.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts.

[Holland, 1992] Holland, J. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, Massachusetts, second edition.

[K. E. Kinnear, Jr., 1994] K. E. Kinnear, Jr., editor (1994). *Advances in Genetic Programming*. MIT Press.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

[Koza, 1994] Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Pres, Cambridge, Massachusetts.

[Koza et al., 1996] Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors (1996). *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA. MIT Press.

[Langdon, 1996] Langdon, W. B. (1996). A bibliography for genetic programming. In Angeline, P. J. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, chapter B, pages 507–532. MIT Press, Cambridge, MA, USA.

[McKay et al., 1995] McKay, B., Willis, M. J., and Barton, G. W. (1995). Using a tree structured genetic algorithm to perform symbolic regression. In Zalzala, A. M. S., editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, volume 414, pages 487–492, Sheffield, UK. IEE.

[O'Reilly, 1995] O'Reilly, U.-M. (1995). *An Analysis of Genetic Programming*. PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada.

[O'Reilly and Oppacher, 1995] O'Reilly, U.-M. and Oppacher, F. (1995). The troubling aspects of a building block hypothesis for genetic programming. In Whitley, L. D. and Vose, M. D., editors, *Foundations of Genetic Algorithms 3*, pages 73–88, Estes Park, Colorado, USA. Morgan Kaufmann.

[Poli and Langdon, 1997] Poli, R. and Langdon, W. B. (1997). An experimental analysis of schema creation, propagation and disruption in genetic programming. Technical Report CSRP-97-8, University of Birmingham, School of Computer Science. Submitted to ICGA-97.

[Whigham, 1995] Whigham, P. A. (1995). A schema theorem for context-free grammars. In *1995 IEEE Conference on Evolutionary Computation*, volume 1, pages 178–181, Perth, Australia. IEEE Press.

[Whigham, 1996a] Whigham, P. A. (1996a). *Grammatical Bias for Evolutionary Learning*. PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy.

[Whigham, 1996b] Whigham, P. A. (1996b). Search bias, language bias, and genetic programming. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 230–237, Stanford University, CA, USA. MIT Press.

[Whitley, 1993] Whitley, D. (1993). A genetic algorithm tutorial. Technical Report CS-93-103, Department of Computer Science, Colorado State University.