CHAPTER 4

The Troubling Aspects of a Building Block Hypothesis for Genetic Programming

In Chapter 3 we pursued experimental evidence that GP conducted a hierarchical search process. One of the reasons for our conjecture was that a GP equivalent of the GA Building Block Hypothesis perhaps could be assumed to operate in GP because GP is a specialization of a GA.

We precisely define a schema in GP and derive a lower bound on the growth of the expected number of instances of a GP-schema from one generation to the next. Following precedent of the literature for GAs which use fixed length binary strings, we refer to this as the GP Schema Theorem (GPST). We also wish to show that, although a notion of building blocks arises from an interpretation of the GPST, it is questionable whether such building blocks reliably exist throughout the course of a GP run. Finally, we emphasize that, as with GAs that use fixed length binary strings, hypothesizing building block combination requires greater liberty with the interpretation of the Schema Theorem than is justifiable.

Our investigation is motivated by the historical precedent of the Schema Theorem and BBH as an explanation of the search power of GAs [46, 32]. Holland's analyses have been the foundation for more precise explanations (some diverging from a schema-based approach) of GA search behaviour. GA theory for fixed length binary strings promises to be a useful source of analogy for GP because, just like other GAs, it uses the same central algorithm loop which applies the basic evolution-based genetic operators and both act as a "shell" which accepts fitness function and problem encoding as parameters. Other GAs and GP use genetic exchange within the population (crossover) and fitness-based selection (and both have been widely employed with fitness proportional selection). The similarity of operators and central algorithm make it worthwhile to formulate a GP theory along the lines of GA theory for fixed length binary strings.

Some recent experimental [82, 28] and theoretical [32, 92, 38, 120, 2, 3] research has questioned the value of the Schema Theorem and BBH as a description of how the GA searches or as the source of the GA's power. In this chapter we confirm that the GPST and a GP BBH similarly fail to provide an adequate account of GP's search behaviour and that various plausible interpretations of the GPST fail to support a GP BBH. Some reasons for the inadequacy of the interpretations are the same as for GAs which use fixed length binary strings, others pertain more directly to GP, and are due to its representation and crossover operator. We hope, however, that is investigation of how interpretations of the GPST fall short of supporting a GP BBH will provide insights for subsequent improved accounts of GP's search behaviour.

The chapter is organized as follows:

Section 4.1 discusses various options for a schema definition, and a more general definition than the one given in [69] is chosen. We also give formal definitions of GP-schemas and of schema order and defining length.

Section 4.2 presents the GPST as a recurrence relation that expresses the lower bound on expected instances of GP-schemas from one generation to the next.

In Section 4.3 we discuss the approximations and questionable assumptions involved in interpreting the GPST to hypothesize that a building block process characterizes GP search.

Section 4.4 concludes the chapter.

4.1. Schema Definition and Related Concepts

The first question to be considered is: what schema definition in GP is useful in formulating a description of GP search? Schemas, or similarity templates, are simply one way of defining subsets of the search space. There are obviously many ways in which the GP search space could be partitioned (e.g. according to function, fitness, number of nodes in tree, height of tree) but it is logical to stay close to the spirit of the GA schema definition because it permits a description of the crossover operator's behaviour to be incorporated into the recurrence relation that counts the schema instances each generation. For example, if we were to instead choose to define subsets of the space according to fitness, we would not be able to explicitly formulate how many instances within a partition of a given fitness would propagate to the next generation since it is not known how crossover affects the samples of this partition.

The first schema definition we will consider is from [69, p. 117–118]. According to Koza,

> a *schema* in GP is the set of all individual trees from the population that contain, as subtrees, one or more specified trees. A schema is a set of LISP S-expressions (i.e., a set of rooted, point-labeled trees with ordered branches) sharing common features.

The distinctive aspect of Koza's schema definition is that a schema is a number of S-expressions each of which is isomorphic to a tree. See Tree A of Figure 48. It can be parsed inorder to form the S- Expression (IF (< 3 4) (+ 1 2) (dec x)) where IF, <, +, and dec are names of S-expressions with 3, 2, 2 and 1 argument, respectively.

Koza's definition implies that no schema is defined by an incompletely specified S-expression such as (+ # 2) where "#" is a wildcard denoting the substitution of any S-expression. There are wildcards implicit in the Koza definition but they are restricted to S-expressions which enclose the schema rather than lie within it. Thus, the schema can also be written correctly as (# (IF (< 3 4) (+ 1 2) (dec x))), with the interpretation that the wildcard can be matched by any S-expression which has at least one argument matching the schema. In other words, the schema can be embedded as a subtree anywhere in a larger tree. In consideration of the variable length representation in GP a wildcard can also be null (i.e. represent nothing and the parentheses which match it are eliminated). In this case the schema defines a partition of one instance.

There are less restrictive schema definitions which are worthy of consideration. First, while this schema definition seems intuitive because subtrees or syntactically complete S-expressions are swapped by GP crossover, it ignores the possibility of an incompletely specified S-expression such as (+ # b) or (+ (# 3 4) b). This sort of S-expression has a specific name (or root in the corresponding tree, + in this example) but some parts within it are not specified because of the presence of internal wildcards. One obvious example of such an incompletely specified S-expression is the part of the parent tree “left behind” to be joined with a subtree taken from the other parent. We call the hierarchical structure (which is not strictly a tree) corresponding to what is left intact by repeated crossovers a **tree fragment** or simply a **fragment**. An example is Fragment B of Figure 48 which corresponds to a schema (# (IF (< # #) (+ 1 #) (dec #))).

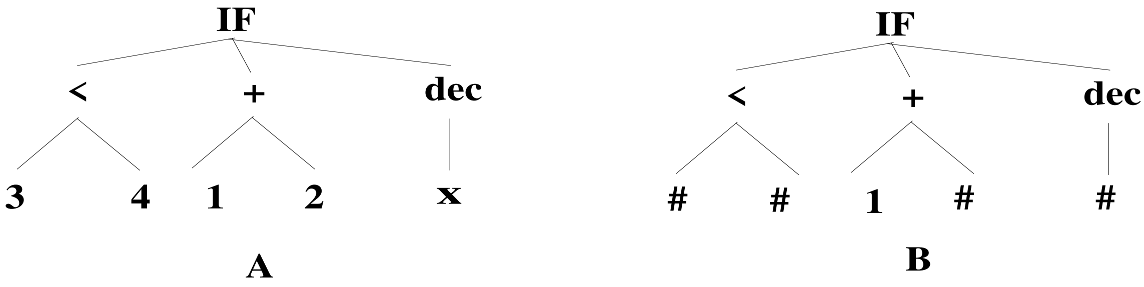

Figure 48. GP-schemas: Tree *(A)* versus Fragment *(B)*. A is a tree because it can be parsed inorder to form a syntactically complete S-expression. All of A’s leaves are variables, constants or primitives that do not require arguments. B is a fragment, not a tree, because it has wildcards as leaves.

A fragment is essentially a tree that has at least one leaf that is a wildcard. It corresponds to an incomplete S-expression with wildcards inside it. There is always a wildcard at the root of a fragment to denote that it can be fully embedded in a tree.

It should be noted that the root wildcard (implicit in Koza’s schema definition) can be matched more freely than a fragment’s leaf wildcard. Although both kinds of wildcard eventually match with a primitive, a primitive can match a leaf wildcard only if it is in a specified position of an argument list. A primitive can match the root wildcard each time one of its arguments matches the specified part of the schema. This is because the schema definition does not state what position the specified part has in the root primitive’s argument list. For example, both (– (+ 3 4) 5) and (– 5 (+ 3 4)) are instances of the schema (# (+ 3 4)) because the definition does not state which argument (+ 3 4) must match. The schema definition is not restricted so as to require a specific argument position, such as the first, or the n-th, to match because, in order to designate the match, wildcards of different arity would have to be introduced. This, in turn, would defeat the generality a wildcard is supposed to provide. Instead this ambiguity is accepted as a natural consequence of a representation which does not use fixed positioning.

Considering fragments, an even more general schema definition is possible. A schema may be defined as an unordered collection of both completely defined S-expressions (as in Koza's definition) and incompletely defined S-expressions (i.e., fragments). The schema definition does not specify exactly how the fragments or S-expressions are linked within an instance but it requires that they must all be matched.

For example, consider the unordered 3 element collection of (+ 3 4), (+ 3 4) and (− # #) and three individuals:

1. (IF (+ 3 4) (+ 3 4) (− x 2)),
2. (IF (− x 2) (+ 3 4) (+ 3 4)) and
3. (AND (+ 3 4) (+ 3 4) (+ 3 4) (− x y))

Individuals 1 and 2 instantiate the schema once and individual 3 actually instantiates it three times because there are three combinations of two (+ 3 4) subtrees and one (− # #) fragment. Figure 49 shows a different example in terms of trees.

This general schema can be written more simply as a set (i.e. an unordered collection without duplicates) of pairs by pairing each fragment or completely defined S-expression with the number of its occurrences that must be matched by an instance, and the root wildcard is implicitly assumed. In the above example, the schema would be represented as {((+ 3 4),2), (− # #),1)}.

The more general schema definition is relevant because it allows the description of partial solutions in GP, i.e., of combinations that persist for more than one generation as crossover dissects and substitutes the parts of a tree corresponding to wildcards. Another reason is this: Consider that GP estimates the fitness of a schema by sampling the fitness of its instantiations (i.e. by sampling the fitness of a program each time the schema is found in it). The accuracy of the estimate depends on how many different samples GP processes. Other considerations being equal, the accuracy of the estimate is not related to whether the schema is a complete S-expression tree or multiple fragments and trees.

We therefore choose to define GP-schemas as follows:

> A **GP-schema** H is a set of pairs. Each pair is a unique S-expression tree or fragment (i.e., incomplete S-expression tree with some leaves as wildcards) and a corresponding integer that specifies how many instances of the S-expression tree or fragment comprise H.

An individual program in the population **instantiates** a GP-schema once **for each way** it matches the number of occurrences of trees and fragments in the GP-schema. For example, in Program ii of Figure 49 schema $H = \{((+\ 3\ 4), 2)\}$ is matched by six different combinations of two subtrees and, thus, Program ii instantiates H six times. While argument position is not considered when placing a fragment into the parse tree, the order of the arguments found inside a fragment must match the order of the arguments in the target subtree. Thus the program (+ 4 3) does not match with the schema $\{((+\ 3\ 4), 1)\}$.

We stress that a program instantiates a GP-schema once **for each way** it matches the GP-schema because we are ultimately interested in counting the expected occurences of a program pattern. Our GP-Schema definition captures the notion of a program pattern and, due to GP's representation, a program pattern (i.e., GP-schema) can occur more than once in a program. Consider a program h. An **instantiation** of a schema H by a program h, $inst(h, H)$, is an element of $Inst(h, H)$, which, in turn, is a function that produces a set of all matches of H by the target program h, based on the matching procedure described above. Figure 49 shows 6 instantiations of $H = \{((+\ 3\ 4), 2)\}$ by Program ii. Though an instantiation designates one specific match of the GP- schema, the notation $inst(h, H)$ does not specify the designation. The designation is, however, important and should be implicitly noted because it is later required in counting the edges connecting the GP-schema instantiation to determine its *defining length*.

Once a GP-schema definition has been adopted, the next task - in analogy to GA theory for fixed length binary strings - is to define measures of GP-schema specificity or order and of GP-schema defining length. These two concepts are used together - again in analogy to GA theory for fixed length binary strings - to determine the likelihood that a GP-schema will be disrupted by crossover. The notion of order makes it possible to compare the relative sample sizes of schemas, and can be transferred directly from GA theory for fixed length binary strings. The notion of defining length, however, can not be directly lifted from the GA domain because of the variable structure of trees and fragments.

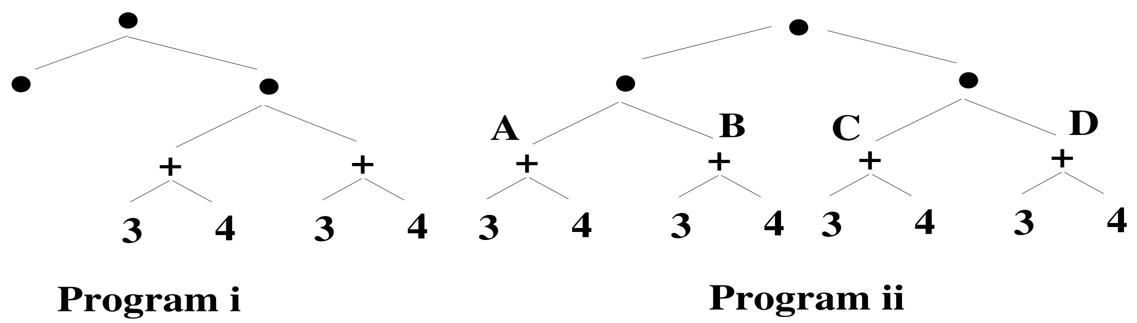

Figure 49. The GP-schema $H = \{((+34), 2)\}$ has one instance in Program i and six instances in Program ii (AB, AC, AD, BC, BD, CD). The order of H is 6 and $D_{fixed}(H) = 4$. In Program i $D_{var}(h) = 2$. The instances in Program ii (see previous list) have $D_{var}(h) = 2, 4, 4, 4, 4, 2$ respectively.

The **order** of a GP-schema is the number of nodes in the graphs corresponding to its S-expressions and fragments. For example, in Figure 49 the schema $H = \{((+\ 3\ 4), 2)\}$ has order 6, the schema $\{((\text{IF}\ \#\ \#\ \#), 1)\}$ has order 1, and the schema $\{((\text{IF}\ a\ \#\ \#), 1)\}$ has order 2.

Order is a straight-forward concept for GP-schemas. Schemas of higher order or greater specificity, other considerations being equal, will have fewer instances in a population than those of lower order or lesser specificity.

The **defining length D** of a GP-schema instantiation is the sum of its variable and fixed defining lengths:

$$D(inst(h, H), H) = D_{fixed}(H) + D_{var}(inst(h, H), H)$$

Below, we let $D(h, H)$ be short for $D(inst(h, H), H)$, and $D_{var}(h, H)$ be short for $D_{var}(inst(h, H), H)$.

The **fixed defining length** of a GP-schema H, $D_{fixed}(H)$, is the number of edges within each S-expression or fragment of H, not including edges connected to a wildcard. It is derivable from the GP-schema alone, independently of any program. For example, in schema $H = \{((+\ 3\ 4), 2)\}$ of Figure 49, $D_{fixed}(H)$ equals 4.

The **variable defining length** of a GP-schema instantiation, $D_{var}(h, H)$, is the number of edges which connect together the S-expressions or fragments in H, i.e., the sum of the lengths of the shortest paths between a schema fragment and the deepest common ancestor of all the schema fragments in the instantiation. $D_{var}(h, H)$ must be calculated for each instantiation and depends upon how the schema instance is embedded in the program. For example, in Program ii in Figure 49, instantiations AB and CD have a variable defining length of 2 and the others have a variable defining length of 4.

Disruption of a GP-schema instantiation $inst(h, H)$ occurs when a node in h is selected as a crossover point and the swapping of the subtree rooted at the crossover point with a subtree from another program changes h sufficiently so that it no longer instantiates H.

Let the sample space of disruption of a GP-schema instantiation be the number of nodes in program h, $Size(h)$, and let $P_d(h, H)$ be short for $P_d(inst(h, H), H)$. The upper bound on the probability of disruption under crossover of a GP-schema instantiation is its defining length divided by the number of nodes in h:

$$P_d(h, H) = \frac{D_{fixed}(H) + D_{var}(h, H)}{Size(h)}$$

Proof: The defining length of a GP-schema instantiation equals the number of crossover events that can destroy it. $Size(h)$ is the number of available crossover locations. In the worst case, no subtree swapped in will re-create the instantiation. □

For example, when $h =$ Program i in Figure 49 and $H = \{((+\ 3\ 4), 2)\}$, $P_d(h, H) = \frac{2}{3}$ since $D(h, H) = 6$ ($D_{var}(h, H) = 2$, $D_{fixed}(H) = 4$) and the total program consists of 9 nodes.

Most reported experiments of GP are run with a probabilistic bias in the crossover point selection. Leaf crossover points are probabilistically selected with a leaf bias $L_b = 0.1$ and interior points are probabilistically selected with bias $1 - L_b = 0.9$. Let the number of leaf nodes in a schema H be $V(H)$. Since the changing of a leaf node by crossover is accounted for by $D_{fixed}(H)$ and not $D_{var}(h, H)$, the probability of disruption can now be biased accordingly to yield the more precise formulation of $P_d(h, H)$:

$$P_d(h, H) = \frac{L_b V(H) + (1 - L_b)(D(h, H) - V(H))}{Size(h)}$$

Proof: The probability that a leaf in schema H will be chosen as a crossover point is $L_b V(H)$. The probability that an interior point in schema H will be chosen is $(1 - L_b)(D(h, H) - V(H))$. The space of all crossover events is $Size(h)$. $\Box$

For example, using again Program i in Figure 49 as h, $P_d(h, H) = 0.24$ since $L_b V(H) = 0.4$ ($L_b = 0.1$ and $V(H) = 4$) and the program consists of 9 nodes.

We define the **compactness** of a GP-schema instantiation as the converse of its probability of disruption: $c(h)$ equals $1 - P_d(h, H)$. Thus, when the probability of schema disruption of an instantiation is high, its compactness is low, and, when the probability of disruption of an instantiation is low, its compactness is high. Since $P_d(h, H)$ is a upper bound on the probability of disruption, $c(h, H)$ is the lower bound on the compactness.

In summary, in this section we have motivated and introduced a general GP-schema definition and defined concepts of specificity, disruption and compactness relating to this definition.

4.2. A GP Schema Theorem

This section formulates a GP Schema Theorem (GPST) that expresses the lower bound of the growth of the expected number of instances of a GP-Schema.

Recall that the GA Schema Theorem for fixed length binary strings expresses the lower bound on growth in expected membership of a schema in the population from time t to $t + 1$. It has three factors:

1. the expected membership of the schema at time t in a population of n strings,
2. the reproductive factor of the expected membership of a schema contributed by fitness proportional selection, and
3. the lower bound estimate of whether a schema member will survive crossover and mutation.

To formulate a similar lower bound for GP the following adjustments are required.

1. The GA Schema Theorem refers to expected **members** of a schema. Membership is appropriate in GAs that use fixed length binary strings because a string instantiates a given schema at most once. The GPST needs to count

the expected **instances** of a GP-schema because a program may instantiate a given schema more than once (see Section 4.1). Let $i(H,t)$ be the number of instances (by virtue of instantiation) of schema H at time t in the population of programs. $E[i(H,t)]$ denotes the expected number of instances of schema H.

2. The expected number of instances of a schema H that are reproduced via fitness proportional selection of programs must be calculated. Let $\hat{f}(H,t)$ denote the estimated average fitness of schema H at time t.

Lemma 1 $E[i(H,t+1)] = i(H,t)\frac{\hat{f}(H,t)}{\bar{f}(t)}$.

Proof: Let n denote the population size and $\bar{f}(t)$ the average fitness of all programs in the population. Let $f(j)$ denote the fitness of program j. Since the number of times a program h instantiates schema H can be calculated as $|Inst(h,H)|$, the number of instantiations of H in the population at generation t, $i(H,t)$, is given by $i(H,t) = \sum_k |Inst(k,H)|$. Fitness proportional selection reproduces a program k with probability:

$$\frac{f(k)}{\sum_{j=1}^{n} f(j)}$$

The expected number of times that program k will be copied into the next generation is:

$$n\frac{f(k)}{\sum_{j=1}^{n} f(j)}$$

and the number of instantiations of H that will be in the next generation because of program k is:

$$n\frac{f(k)}{\sum_{j=1}^{n} f(j)}|Inst(k,H)|$$

Thus, the expected total number, $E[i(H,t+1)]$, of instantiations of H in the next generation, is:

$$E[i(H,t+1)] = \sum_{k=1}^{n} n\frac{f(k)}{\sum_{j=1}^{n} f(j)}|Inst(k,H)|$$

Since $\bar{f}(t) = \frac{\sum_{j=1}^{n} f(j)}{n}$,

$$E[i(H,t+1)] = \frac{1}{\bar{f}(t)}\sum_{k=1}^{n} f(k)|Inst(k,H)|$$

$$E[i(H,t+1)] = \frac{i(H,t)}{\bar{f}(t)}\sum_{k=1}^{n} \frac{f(k)|Inst(k,H)|}{I(H,t)}$$

Since the estimated average fitness is:

$$\hat{f}(H,t) = \sum_{k=1}^{n} \frac{f(k)|Inst(k,H)|}{I(H,t)}$$

Therefore, $E[i(H,t+1)] = i(H,t)\frac{\hat{f}(H,t)}{\bar{f}(t)}$ □

To see how a schema that appears several times in a program can be expected to reproduce in proportion to its fitness via fitness proportional selection on programs, it is important to realize that the average fitness of a schema H is the sum of the partial contributions of the fitnesses of programs that instantiate H. The fitness contribution of a program is proportional to the number of instantiations the program adds to the total instantiations of H in the population. If the above divisor were the number of programs that are members of the schema, rather than $i(H,t)$, fitness proportional selection would not guarantee expected reproduction relative to estimated average fitness.

3. Because the standard GP [69] does not use mutation, we do not account for it. We define an estimate of the probability of disruption of any instantiation of H because $P_d(h,H)$ is not the same for every GP-instantiation involving H. The upper bound probability of disruption of any instantiation of H is defined as follows:

$$P_d(H,t) = \sup P_d(h,H) \text{ at time t.}$$

$P_d(H,t)$ is a very conservative upper bound. The GPST would be more precisely bounded if the probability of disruption of schema instances were more accurately represented.

The modifications result in the following GP Schema Theorem, where $E[i(H,t)]$ denotes the expected number of instances, not members, of H at t, $\hat{f}(H,t)$ is the observed average fitness of the instances of the GP-schema H, $\bar{f}(t)$ is the average fitness of the population, $P_d(H,t)$ is the upper bound on the probability of schema disruption, and P_{xo} is the probability of using crossover:

$$\textbf{GP SCHEMA THEOREM:} \quad E[i(H,t{+}1)] \geq i(H,t)\frac{\hat{f}(H)}{\bar{f}(t)}(1-P_{xo}P_d(H,t))$$

Proof: By Lemma 1, without crossover, the expected reproduction of schema H is $i(H,t)\frac{\hat{f}(H,t)}{\bar{f}(t)}$. With crossover, the lower bound probability that schema H survives intact is $(1-P_{xo}P_d(H,t))$ because $P_d(H,t)$ is an upper bound on the probability that a schema H will be disrupted. $\square$

On first glance, the above recurrence is the same as the GA Schema Theorem but an interesting difference is due to the "**crossover survival term**" $(1-\ P_{xo}P_d(H,t))$. This is an **estimate** of the minimal number of schema instances that survive crossover. As in GAs that use fixed length binary strings, the **actual** minimum likelihood of survival may be greater than the estimate and may change each generation. Whether this is indeed the case depends upon the composition of the population from which mates are drawn. Given an appropriate subtree from its mate, a schema disrupted by removing a subtree at the crossover point can be "repaired" and reinstantiated. The estimate of crossover survival in both accounts is inaccurate because it does not account for such an event.

However, in GP, the inaccuracy of the crossover survival term is further exacerbated by the fact that both the size and shape of a program containing a schema instance can change in a generation even when the schema instance is not disrupted. As a consequence of the variable length representation and the behaviour of the crossover operator, the defining length of a schema and the size of the program in which it is embedded both can change. These changes will also impact the observed probability of disruption.

Not only is crossover probabilistic, but fitness proportional selection is not explicitly correlated to program size and height[1]. These facts, together with the facts stated in the previous paragraph, imply that the probability of disruption of a schema

[1]Fitness proportional selection is probably implicitly correlated to program size and height but the relationship is not known and it may differ for each GP problem.

- while an upper bound can be formulated - changes so drastically from generation to generation, and in such an unpredictable fashion, that it can best be represented as a random variable. Since the term $P_d(H,t)$ in the GPST can be considered a random variable, we refer to it as $\tilde{D_t}$.

$\tilde{D_t}$ represents the observed probability of disruption of schema H in the population at time t. We use $\tilde{D_t}$ to formulate definitions of disruption and compactness. This allows us to interpret the GPST with respect to the allocation of trials among schemas.

Schema Disruption: Let R be the event that, at time t, $\tilde{D_t}$ is less than β, a constant. The disruption likelihood of a schema H is defined as the probability of event R, P_R. For $P_R < \alpha$, a constant, a schema is disruption prone.

Schema Compactness: Compactness is defined as $1-P_R$. If, at time t, $P_R > \alpha$, a constant, a schema is compact. Intuitively a schema is compact if its maximum probability of disruption is low regardless of the size and structure of the programs which contain it.

4.3. Building Block Definition and Building Block Hypothesis

In this section we propose and critically examine a definition of GP building blocks and a GP Building Block Hypothesis (BBH). Both the definition and the hypothesis result from an interpretation of the GPST and are intended to be fully analogous to the definition of building blocks for GAs that use fixed length binary strings and to the BBH for GAs that use fixed length binary strings. However, as will be pointed out below, our seemingly straightforward interpretation of the GPST rests on several questionable assumptions. Without these assumptions, no GP BBH can be formulated in analogy with the GA BBH.

GP building blocks: GP building blocks are low order, consistently compact GP schemas with consistently above average observed performance that are expected to be sampled at increasing or exponential rates in future generations.

GP Building Block Hypothesis (BBH): The GP BBH states that GP combines building blocks, the low order, compact highly fit partial solutions of past samplings, to compose individuals which, over generations, improve in fitness.

Thus, the source of GP's power, (i.e., when it works), lies in the fact that selection and crossover guide GP towards the evolution of improved solutions by discovering, promoting and combining building blocks.

Let us now review the assumptions presupposed by the GP BBH.

1. The GP BBH refers to the combining of schemas yet the GPST, by referring to the expected instances of only one schema, fails to describe the interactions of schemas. In this respect, the GP BBH is not supported by any interpretation of the GPST.

   Previous GA work in the context of fixed length binary strings [39, 77, 122] has made this point in much more detail. Many complicated interactions between competing schemas and hyperplanes take place in the course of a GA run. None of this activity can be described by a Schema Theorem because the latter simply considers one schema in isolation. Since the GPST does not differ from the GA Schema Theorem in this respect, the above argument applies with equal strength to GP.

   Vose has pointed out that, without knowing the composition of the population in a GA, it is impossible to precisely state how schemas combine and how many schemas can be expected [120]. Again, this point applies equally to GP.

2. The GPST also fails to lend support to the GP BBH because hyperplane competition in GP is not well defined. In GAs, trial allocation competition takes place among hyperplanes which have common features but where each "competitor" differs in the expression of that feature. The lack of a feature-expression orientation in the GP representation (i.e., GP's non-homologous nature) results in an unclear notion of which hyperplanes compete for trial allocation. This inherent lack of clarity concerning hyperplane competition seems to indicate that schema processing may not be the best abstraction with which to analyze GP behaviour.

3. Grefenstette [38] has called the classic GA BBH a "Static Building Block Hypothesis". This he states as

> Given any short, low order hyperplane partition, a GA is expected to converge to the hyperplane with the best static fitness (the "expected winner"). [38, p. 78]

Static fitness is defined as the average of every schema instance in the entire search space to distinguish it from the observed fitnesses the GA uses as an estimate of static schema fitness. He argues that "the dynamic behavior of a GA cannot in general be predicted on the basis of static analysis of hyperplanes" [38, p. 76]. Two of the reasons that the true dynamics of a GA is not estimated by the static fitness of schemas are "collateral convergence" and high fitness variance within schemas. The first reason is that, once the population begins to converge even a little, it becomes impossible to estimate static fitness using the information present in the current population. The second reason is that, in populations of realistic size, high fitness variance within schemas, even in the initial generation, can cause the estimate and static fitness to become uncorrelated.

This argument applies to GP. Furthermore, the issue of high fitness variance within a schema may be especially important in GP. As far as we know, the amount of fitness variance for GP schemas has not been empirically sampled. To discuss the issue, one must consider that schemas in GP are "pieces of program". A schema instance acquires the fitness of the program which embeds it. If the primitives are functionally relatively insensitive to context, there may be schemas in the search space that are also relatively insensitive to program embedding and thus have low fitness variance among their instances. GP is also known to evolve large programs full of functionally inert material. This material may act to shield partial solutions from interference with each other and prevent their fitness from changing when surrounding code is sampled [2]. In contrast, it intuitively seems that rearranging code or simply inserting a new statement into a program can lead to drastic changes in its fitness. This argues that the fitness variance of a schema's instances may be high.

[2] Admittedly this is simplistic; in programs it is difficult to ever clearly state that pieces of code do not interact.

4. The assumption of expected increasing or exponential trials for building blocks requires certain behaviour to be constant over more than one time step. The GPST does not describe behaviour for more than one time step and it is not the case that the required behaviour is constant.

   The inaccuracy in the assumption arises from estimating the long term behaviour of the reproduction and crossover survival terms in the GPST. In fact, the GPST describes behaviour for only one step and this hides important dependencies in the iteration [3]. The GPST states that **in the next generation** schema H grows or decays depending upon a multiplication factor that is the product of two terms: the probability of the schema being reproduced (i.e. the schema's fitness relative to the population average) and the probability that the schema is not disrupted.

   $$\textbf{MultiplicationFactor} : \frac{\hat{f}(H,t)}{\bar{f}(t)}(1 - P_{xo}P_d(H,t)) \tag{4.1}$$

   Clearly if the Multiplication Factor (4.1) is greater or equal to 1, the expected trials of a schema will increase **in the next generation**.

   Interpreting the GPST to describe the expected allocation of trials to a schema **asymptotically** or **over more than one generation** relies on interpreting the Multiplication Factor in the GPST for more than one time step. If the time dependence of the terms were ignored by assuming that the margin by which a schema's estimated average fitness is better than the population average is constant and that $P_d(H,t)$ never changes, the claim of expected **exponentially** increasing trial allocation would be justified.

   If we resist ignoring the time dependence of the two terms (because they are a reality!) to avoid misleading over-simplification, the assumption that the expected number of trials will grow exponentially is weakened by the qualification that the Multiplication Factor must be stationary:

   $$\frac{\hat{f}(H,t)}{\bar{f}(t)}(1 - P_{xo}P_d(H,t)) = \frac{\hat{f}(H,t+1)}{\bar{f}(t+1)}(1 - P_{xo}P_d(H,t+1))$$

[3] See [3] for a different and crucial Schema Theorem dependency

and the crucial time dependent relationship is the logarithmic growth of the reproductive term relative to the negative logarithmic growth of the crossover survival term:

$$\Delta \log \frac{\hat{f}(H,t)}{\bar{f}(t)} = -\Delta \log \left(1 - P_{xo} P_d(H,t)\right)$$

5. As a schema starts dominating, the margin by which it is fitter than the average fitness of the population decreases. The only thing that enables it to continue growing is a decrease in its probability of disruption. The problem is that there is no guarantee that $\tilde{D}_t$ decreases at a rate ensuring positive growth of expected allocation of trials over the same interval.

   We can only consider the plausibility of this decrease in the upper bound on disruption likelihood in this situation. An exact answer is not possible[4]. Consider the growth of programs in GP runs: In GP the maximum height or size of a program is set to a lower value for the initial population than the value crossover is constrained to use in creating trees in subsequent generations. This allows programs to grow larger each generation (up to the maximum). We cannot make precise statements about the size and height distribution of a schema's instances but if one assumes they are uniformly distributed within the population, program growth in each generation may indeed cause the upper bound on the probability of disruption to decrease. Whatever the circumstances of program growth, in GP any decrease in the likelihood of disruption is fortuitous or roundabout rather than explicit in the algorithm. That is, the crossover operator and selection process do not explicitly control the size and shape of programs in a correlation with fit schemas. It should be noted that the decrease in probability of disruption caused by program growth also works in favor of unfit program fragments. Because of the variable length representation of GP, the 'cushioning' of unfit programs due to program growth is more of a problem than it would be in GAs that use fixed length binary strings.

6. Building blocks may only exist for a time interval of the GP run because the

[4]Because it requires an account of population composition that is lacking in the GPST.

estimated fitness relative to the population fitness and upper bound probability of disruption of a schema vary with time.

Consider an interpretation of the GPST that is intended to explain why a particular GP run did not find an optimal solution. An explanatory hypothesis might be that consistently highly fit partial solutions are not consistently compact. Or, that consistently compact partial solutions are not highly fit. These hypotheses reveal a caveat of the BBH : a partial solution is a building block only if its sample is consistently **both** above the population average in fitness and compact (i.e., consistently has a low maximum probability of disruption). When the Multiplication Factor is not greater than one, **despite fit partial solutions or compact partial solutions**, building blocks **do not exist**.

To elaborate further, consider an interval when a schema's margin of fitness above the population stays constant. This could happen when the observed average fitness of the schema increases (due to updated sampling) at the same rate as the population fitness. In this interval, the upper bound probability of the schema's disruption becomes the crucial factor in determining whether it will be a building block. Relative to its fitness, the schema could be allocated fewer trials for one interval than a comparable schema because the tree sizes and shapes of its instances have changed. This implies that, in some sense, **partial solutions can be inert as building blocks at some generations and active at others**. It is not a conceded fact that a building block persists in the subsequent course of the run. Indeed, in the described circumstance highly fit partial solutions may never be building blocks because, despite reproduction, they could be too prone to disruption.

7. The BBH assumes that solutions can be arrived at through linear combination of highly fit partial solutions. This is a statement about the problem of program induction rather than GP. There is no basis for assuming that a solution's subcomponents are independent. The BBH is a statement about how GP works only if there is linearity in the solution.

The basic lesson is that the GPST (and any similar schema theorem) omits important dependencies from the recurrence and is, thus, bound to oversimplify GP dynamics. In particular, the dynamics of crossover and selection that are of interest last longer than one time step. The BBH also assumes the existence of the same building blocks throughout a run and is not specific about the dynamics of building block discovery, promotion and combination in the course of a run.

In summary of Section 4.3, we presented a definition of GP building blocks and a GP Building Block Hypothesis. We then discussed crucial issues in their usefulness and credibility. The most serious issue concerns the time dependent behaviour of schema disruption and observed average fitness relative to the population fitness. We also have cautioned that there are times when the BBH will not hold because the BBH presupposes the existence of building blocks despite the fact that compactness and consistent fitness are not guaranteed.

4.4. Conclusion

We conclude that the GP BBH is not forthcoming without untenable assumptions. Our critical discussion has led us to identify what we take to be perhaps the major problem of GP: it exerts no control over program size but program size directly affects disruption probability. Furthermore, how the probability of disruption of a schema changes over time, even from one generation to the next, is unpredictable. This time dependent behaviour is almost certainly a stochastic process (i.e., it may have underlying structure but is primarily driven by randomness): while selection and crossover determine the structure of individuals for the next generation, they control program size - which affects disruption probability - in only a roundabout way. A more useful and precise GP building block definition should state something about the time dependent behaviour of the probability of disruption but this is not quantifiable without some empirical data or simulation.

4.5. Summary

In this chapter we carefully formulated a Schema Theorem for GP using a schema definition that accounts for the variable length and the non-homologous nature of GP's representation. In a manner similar to early GA research, we used interpretations of our GP Schema Theorem to obtain a GP Building Block definition and to state a "classical" Building Block Hypothesis (BBH): that GP searches by hierarchically combining building blocks. We report that this approach is not convincing for several reasons: it is difficult to find support for the promotion and combination of building blocks solely by rigourous interpretation of a GP Schema Theorem; even if there were such support for a BBH, it is empirically questionable whether building blocks always exist because partial solutions of consistently above average fitness and resilience to disruption are not assured; also, a BBH constitutes a narrow and imprecise account of GP search behaviour.

Other related work conducted into formulating an understanding of GP based upon schema or building block processing has been proposed ([90]) and conducted ([115]).

The doubts raised in this chapter about the GP BBH strengthen our conclusion in Chapter 3, that was based upon experimental investigation. It is unlikely that GP conducts its search exploiting true hierarchical process.

Furthermore, the formal investigation of this chapter fails to support any claim that the evolution-based search process of GP plays an exclusive role in the success of program discovery. It suggests that other search algorithms may also succeed in discovering problems described in the program discovery framework. It is in this direction we shall now proceed. Rather than assuming that GP is better than other techniques on the general grounds that it accumulates good partial solutions in parallel and hierarchically combines them, we shall empirically investigate other search algorithms and compare them to GP.