Using Building Block Functions to Investigate a Building Block Hypothesis for Genetic Programming

Una-May O'Reilly
Franz Oppacher

SFI WORKING PAPER: 1994-04-020

SFI Working Papers contain accounts of scientific work of the author(s) and do not necessarily represent the views of the Santa Fe Institute. We accept papers intended for publication in peer-reviewed journals or proceedings volumes, but not papers that have already appeared in print. Except for papers by our external faculty, papers must be based on work done at SFI, inspired by an invited visit to or collaboration at SFI, or funded by an SFI grant.

©NOTICE: This working paper is included by permission of the contributing author(s) as a means to ensure timely distribution of the scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the author(s). It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may be reposted only with the explicit permission of the copyright holder.

www.santafe.edu
Using Building Block Functions to Investigate a Building Block Hypothesis for Genetic Programming

Una-May O'Reilly
Santa Fe Institute
1660 Old Pecos Trail, Suite A
Santa Fe, NM, 87505
unamay@santafe.edu

Franz Oppacher
School of Computer Science
Carleton University
Ottawa, Ont, CANADA, K1S 5B6
oppacher@scss.carleton.ca

Revision Date: Wed, April 13, 1994.

Abstract:

This paper presents building block functions, i.e., functions in which explicit schemas of high fitness are defined (BB functions, for short) which are useful in investigating the character of Genetic Programming (GP) search. One conjecture we believe to be answerable through experimentation with these functions is whether GP exploits building blocks (see [O'Reilly and Oppacher 1994]). That is, is one explanation for GP's power that, when primary partial solutions are discovered, their numbers increase and GP crossover is able to combine them into increasingly larger sub-solutions and eventually find the solution? The functions should also provide insight into more detailed aspects of the roles of GP crossover and GP genotype growth.

Introduction

One current approach to investigating the details of GA search is to study its behaviour on functions which are hand designed for particular landscape features [Goldberg 1987; Goldberg 1989a; Liepins and Vose 1990; Whitley 1990; Mitchell, Forrest et al. 1991; Forrest and Mitchell 1992]. This research primarily focuses upon understanding what stages the search may have (e.g. building block discovery, building block combination), interpreting the role of the genetic operators and forming explanations of how the GA converges. We propose that it is timely to pursue similar goals with Genetic Programming (GP) [Koza 1992].

For GAs, a set of functions [Mitchell, Forrest et al. 1991; Holland 1993] has been designed to verify, in more detail, whether the combination of building blocks does occur according to the Building Block Hypothesis [Goldberg 1989b; Holland 1992]. In these functions the fitness of an individual is defined to explicitly reflect the building blocks it contains. These functions were named "Royal Roads" because they potentially map out for the GA the path to the optimum by designating building blocks within the perfect individual and conferring credit to programs which hold them. These so-called "Royal Road" functions are extremely valuable because they facilitate controlled variation of the fitness landscape which allows detailed insights into search behaviour. Performance comparison amongst variations is straightforward because all use the same perfect individual and all possible fitness values are known to the same degree in advance. Perhaps, most importantly "all of the desired schemas are
known in advance, since they are explicitly built into the function. Therefore, the dynamics of the search process can be studied in detail by tracing the ontogenies of individual schemas" [Forrest and Mitchell 1992].

This paper presents Building Block (BB) functions which may help discover potential Royal Roads in GP search. They have been designed to handle the aspects of GP which are distinct from GAs (e.g. variable length genotypes). We have previously formulated in some detail a Schema Theorem, a building block definition and a Building Block Hypothesis for GP (see [O'Reilly and Oppacher 1994]). We will show why experimentation with these functions may yield an answer as to whether GP exploits building blocks by the crossover operator combining them into increasingly larger sub-solutions. We will also suggest other productive experiments.

The paper proceeds in the following manner: Section 1 succinctly defines a GP building block and outlines the GP Building Block Hypothesis. See [O'Reilly and Oppacher 1994] for complete details. Section 2 explains the design features of BB functions for GP. Section 3 shows how two different BB functions can help investigate the validity of the GP Building Block Hypothesis. Section 4 presents variations of these functions for other investigative purposes. We conclude by urging that conjectures about GP search concerning building block processing be validated with this sort of experimentation.

Section 1: GP Building Blocks and a GP Building Block Hypothesis

See Figure 1 for examples of a GP schema and schema instances within programs. A GP schema \( H \) is defined by one or more S-expression trees or fragments (a fragment corresponds to a tree of primitives where leaves do not necessarily have to be primitives of zero arguments) each with a corresponding integer that specifies how many instances of it are in the definition. 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 definition. For example, in Program II of Figure 1 schema \( H = (((+ 3 4), 2)) \) is matched by six different combinations of two subtrees and, thus, Program II instantiates \( H \) six times. The pair consisting of a program \( h' \) and an instance of a schema \( H \) within it are an instantiation. Note that no node corresponding to a primitive may be used more than once in a schema instance but the set of instances of a schema in a program could possibly refer to the same node more than once.

![Figure 1 GP schema Example](image-url)
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 1 the schema \( H = \{(+3,4), 2\} \) has order 6.

The defining length \( D \) of a GP schema instantiation is the sum of its variable and fixed defining lengths: \( D(h',H) = D_{\text{fixed}}(H) + D_{\text{var}}(h') \). The fixed defining length is the number of edges within each S-expression or fragment of \( H \). It is derivable from the GP schema definition alone, independent of a program. For example, in Figure 1 when \( H = \{(+3,4), 2\} \) \( D_{\text{fixed}}(H) \) equals 4. The variable defining length must be calculated for each instantiation and depends upon how the schema instance is embedded in the program. It is the number of edges which connect together the S-expressions or fragments in \( H \). This amounts to the length of the shortest path to a common ancestor. For example, in Figure 1 Program II for the instantiation \( h' \) when subtrees A and D match with schema \( H \), \( D_{\text{var}}(h') \) equals four.

The number of nodes in a program and the defining length of the schema instantiation are the factors in the likelihood of its disruption. The defining length divided by the number of nodes in the program yields \( p_d(h') \), the probability of disruption under crossover. For example, in Figure 1 Program I \( p_d(h') \) equals 2/3 because \( D(h',H) \) equals 6 (\( D_{\text{var}}(h') = 2 \), \( D_{\text{fixed}}(H) = 4 \)) and the program consists of 9 nodes. Compactness \( C(h') \) equals \( 1 - p_d(h') \). Thus when the probability of schema disruption of an instantiation is high, the compactness is low, and, when the probability of disruption of an instantiation is low, compactness is high.

Where \( i(H,t) \) is the number of instantiations of schema \( H \) at time \( t \) in the population of programs, the average probability \( \overline{p}_d(H) \) that a schema will be disrupted by crossover is

\[
\overline{p}_d(H) = \frac{\sum_{h'=1}^{i(H,t)} p_d(h')}{i(H,t)}
\]

The Schema Theorem [Holland 1992] estimates the change in membership of a schema in the population from time \( t \) to \( t+1 \). In consideration of the GP crossover operator and the lack of a mutation operator in the standard GP [Koza 1992], the probability that crossover and mutation will disrupt a schema once it has been reproduced needs to be revised in a Schema Theorem for GP. The average probability that a GP schema \( H \) survives crossover is \( 1 - p_{xo} \overline{p}_d(H) \) where \( p_{xo} \) is the probability of crossover. This results in a GP schema Theorem where \( f(H) \) is the observed fitness of \( H \) and \( \bar{f} \) is the average fitness of the population

**GP Schema Theorem:**

\[
i(H,t+1) \geq i(H,t) \cdot \frac{f(H)}{\bar{f}} [1 - p_{xo} \overline{p}_d(H)]
\]

\( \overline{p}_d(H) \) must be recognized as a random variable in this theorem because the sizes and shapes of programs in a population change each generation without regard to the schema under consideration. Interpreting it as a random variable \( \bar{D} \) yields the definition of compactness for a schema: let \( E \) be the event that \( \bar{D} \) is less than \( \beta \), a constant. Compactness is defined as \( 1 - p_E \). For \( p_E > \alpha \), a constant a
schema is compact. Intuitively a schema is compact if its probability of disruption remains low each generation regardless of the change of the size and structure of programs which contain it.

The effect of the reproduction and crossover terms in the GP Schema Theorem is that schema $H$ grows or decays depending upon a multiplication factor. The factor depends upon two things: whether the schema is consistently above or below the population average in fitness and whether the schema is remains compact. Compact schemas with above average observed performance are GP building blocks. They will be sampled at exponentially increasing rates.

A GP Building Block Hypothesis (BBH) is that GP combines building blocks, the best partial solutions of past samplings, to compose individuals which are of improving fitness. The source of GP's power, (i.e., when it works), is that selection and crossover guide GP in building block discovery, promotion and combination towards the evolution of improved solutions. This conjecture is based on the GP Schema Theorem which implies that compact schemas with above average observed performance are sampled at exponentially increasing rates, and, upon a direct interpretation of the behaviour of the crossover operator.

This GP BBH seems to explain why GP is possibly superior to simpler gradient descent search methods (given that a mutation operator was defined for variable length hierarchical representations) and why crossover is an effective search operator. However, it is imprecise (as is the GA BBH) because of the limitations of the GP Schema Theorem. Truly useful information about GP search behaviour requires characterization at detail greater than the BBH provides. Specifically, a precise and quantitative description of how schema processing takes place is desirable. Optimistically a GP BBH is useful as a high level interpretation from which more interesting and useful questions can be formulated. In GAs this approach combined with experimentation with Royal Road functions has helped to identify overlapping stages of the search and to yield insight into the role of the crossover operator throughout the course of the search [Forrest and Mitchell 1992]. To elaborate upon the GP Building Block Hypothesis in [O’Reilly and Oppacher 1994] we were able to exploit the work previously done with GA’s [Mitchell, Forrest et al. 1991; Forrest and Mitchell 1992]. First, we could pose similar questions about building block interaction and combination and the role of crossover. Second, we could conjecture that the overlapping stages and crossover behaviour which occur in the course of a GA run occur in the same manner in GP. In particular [Forrest and Mitchell 1992] have identified three stages of the search with respect to the role of crossover: Stage 1 is the time it takes for the lowest-order schemas to be discovered, Stage 2 is the time it takes for crossover to combine lower-order schemas into a higher-order schema, and Stage 3 is the time it takes for the higher-order schema to take over the population.

Before we introduce the building block functions we have designed and outline possible experimentation there is a caveat regarding our assumptions at the outset. First, as we have previously noted in [O’Reilly and Oppacher 1994] the BBH explanation for GP’s success is predicated upon the assumption that building blocks are actually present in the search landscape and that the heuristic of
combining them is profitable. BB function investigation will make this assumption but it should be noted that in reality it sometimes does not hold. This is because building blocks or partial solutions which can be combined to larger, fitter solutions are artifacts of a search landscape. A particular landscape arises from the GP engineer’s choice of a particular representation and fitness function for a selected problem. That is, the primitives which have been chosen, the test suite, and the metric which determines the worth of a solution relative to the perfect solution, all together determine whether small fit partial solutions exist and whether they will combine into larger fitter solutions. If BB function investigation confirms the importance of building blocks to GP’s success, it will be advantageous to purposely make choices which create a desirable landscape. This in itself is a difficult issue. The BB functions we will present are “tunable” which means we can model the presumably disadvantageous landscape and investigate GP’s behaviour searching it. At present however, we propose experiments to answer questions of how GP searches using landscapes which have building block properties and thus are presumably easy for it to search.

Thus, in this paper we want to focus on the schema processing behaviour of GP given that building blocks do exist and that the heuristic of improvement through combination is applicable. The objective of this paper is to design for GP a set of problems (which will be specified by primitives and fitness functions) which have idealized and tunable landscape features so that we can propose questions that could be answered by experimentation with them. One important question is what specific landscape features are relevant to good performance in GP? If building blocks of equal fitness and order are considered to be on a “level”, how sensitive is GP to the relative fitness of between levels? Will a non-linear fitness relationship adversely affect either the rate of convergence or fitness of the best solution evolved? Will it simply change the relative duration and overlaps of the stages of the search or will it result in the elimination of a stage or unanticipated behaviour? Is there a range of fitness relationships for which performance is robust or do small variations always have an impact?

It is impossible to a priori precisely designate a building block in GP building block functions because for a schema to have low probability of disruption depends upon which instances of it are in the sample of programs (in terms of sizes and structure) in the population. An important question is: exactly which schemas that confer fitness on programs in the BB functions actually do exhibit compactness? A study of the fitness distribution and disruption probability distribution of schemas over generations to see how it is affected by the bias towards non-leafs nodes in crossover point selection and how it is related to the size and structure distribution of programs in a population is desirable. It would be insightful to ascertain how crucial the distribution of the probability of disruption is. That is, are the probabilities at the center of the disruption distribution important or is the tail of the distribution more crucial?

Additional questions stem from trying to explicitize the role of GP crossover. GP crossover can be seen as alternatively disruptive (because it breaks apart schemas) and combinative (because it moves subtrees among programs). Is the size (i.e. order) of building blocks at successive levels important with
respect to GP's ability to discover all of the blocks of one level and effectively promote and combine them? Does disruption play a useful role in the early course of a run when primary building blocks need to be discovered? How sensitive is GP to how big the lowest-level blocks are and how large successive combinations are? What is the turning point in terms of building block quantity and compactness when the impact of disruption by crossover yields to combination? Is this timing so sensitive as to affect the eventual solution which is found?

Section 2: Building Block (BB) Functions in GP

One way to confirm or reject the GP Building Block Hypothesis, our conjectured stages of GP search, and to answer our questions about its detailed nature is to use functions where schemas are defined a priori and can be explicitly tracked in terms of quantity during the course of a run. We now explain how BB functions can be designed for GP. Succinctly the basic notion is to define some program as perfect then to designate schemas within it as building blocks in the sense that they confer fitness upon programs which contain them.

The search space of a GA function is fixed length bit strings. For the purposes of GP explicit schema-based investigation, without loss of generality, the domain can be considerably simplified if both the primitive alphabet and the shape of the trees isomorphically corresponding to S-expressions in the domain are restricted. This is accomplished by using two primitives \( \{0,1\} \) which have two arguments (nodes with 2 children) and two primitives \( \{0',1'\} \) which have no arguments (leaves). This search space is the set of binary trees with height in the range of 1 to the parameter \( \text{maxHeight} \) where a tree has any combinations of 0s, 1s, 0's and 1's. This search space can be simply modified to be larger or contain trees of a different shape. This would be done by respectively increasing the alphabet or increasing the number of arguments of internal nodes.

The perfect individual is defined a priori in a BB function. In the GA Royal Road functions it is any randomly chosen bit string. Choosing the perfect individual in GP BB functions requires care because building blocks are not fixed to positions in GP. The perfect individual must have distinct subtrees so that the recombination process can be distinguished. This criteria also mimics the likelihood seen in programs that they are composed on unique components. One possible perfect individual is presented in Figure 2. Note that subtrees of the same size are distinct.

![Figure 2: One Possible Perfect Individual in BB Functions](image)

Recall that the idea behind BB functions is that an individual is assigned a fitness corresponding to which building blocks it has instances of. Each building block is assigned a value which it contributes to the fitness of any individual which contains it. This value is meant to indicate that the block is
consistently above average in fitness and it should indicate that the block is compact. Because the compactness of a building block in GP depends upon the manner in which it is embedded in a program (see Figure 3), in a BB function compactness is assessed once a building block is found in an individual. Thus, more accurately, in BB functions schemas in the perfect individual are initially designated as building blocks but the name is correct only to the extent that they confer fitness. The compactness of the designated schemas is not actually known until in the course of a GP run. Since in the course of a run it can be expected that these schemas (somewhat prematurely termed building blocks) will differ in compactness, BB functions “on the fly” confer greater fitness to programs holding compact instances. This serves to stress the BBH in stricter detail: i.e., that building blocks which are consistently above average fitness and which remain compact are the components of the ultimate solution. This is accomplished by another factor in the fitness assessment of a program which accounts for the compactness of an instantiation. Where N is the number of nodes in a program h and D(h',bb) is the defining length of schema bb, compactness \( C_{bb} \) of a building block within a program in a GP BB function is calculated by:

\[
C_{bb} = 1 + \frac{N - D(h',bb)}{N}. \quad 1 < C_{bb} < 2
\]

This is the same as the definition of compactness for a schema member previously given in Section 1 with the constant 1 added so that compactness of a building block will always be between 1 and 2.

In a GA Royal Road function because all genotypes are of fixed length an individual either matches the perfect individual or does not. In GP, by contrast, an individual can be larger than the perfect individual and thus actually match it in addition to being composed of other program primitives. If we consider that such an individual does not match the perfect individual, the fitness calculation for an individual must account for how close its size is to the perfect individual. Therefore, in BB functions the fitness of a program depends not only upon what building blocks it contains but also upon how closely its size matches that of the ideal individual. We term this factor in fitness “size discount".

![Tree A](image1)  ![Tree B](image2)  ![Tree C](image3)  ![Tree D](image4)

Examples of schemas (encircled) which vary in compactness. Trees A and B instantiate the same schema but are of different sizes affecting the compactness. Trees C and D instantiate a multiple element schema implying a variable defining length which also affects the compactness.

Figure 3 Compactness
Where $J$ is the number of nodes the perfect individual, $N$ is the number of nodes in the individual which is being assessed fitness this size discount $S$ is:

$$S = 1 - \frac{\text{J} - \text{N}l}{\text{J}}$$

This ensures that two programs differing from the size of the perfect individual by the same amount but one being smaller and the other being larger are assessed the same "penalty" for incorrect size*. The size discount is another tunable feature of the BB functions because it can be changed if one wants to account for junk genes.

Finally, a BB function computes the fitness of each individual by adding up the value $V$ of each building block $bb$ it contains. In GP this value is the order of the GP schema. Then each value is modified by compactness of the building block. In order to account for the size of a program relative to the size of the perfect individual, the overall fitness is discounted by the ideal size factor. In summary, the fitness of an individual can be stated as: $(\Sigma V_{bb} \cdot C_{bb}) \cdot S$. In Section 3 we provide examples of this calculation.

**Section 3: BB Functions to Investigate the GP Building Block Hypothesis**

Initially we propose two functions, BB1 and BB2, for experiments we name the **GP Building Block** series. These are similar to the first Royal Road functions designed for GAs [Mitchell, Forrest et al. 1991]. In both functions a 15 node full binary tree with distinct subtrees is designated as the perfect individual (see Figure 1). The order of a schema is chosen as its building block value ($C_{bb}$).

In the first function, BB1, all building blocks are trees or fragments of order 3. There are no building blocks which are intermediate combinations of smaller blocks. See Figure 4 where the building blocks are encircled. This designation represents a non-hierarchical path to the optimum because no intermediate combinations of blocks are defined. The defining length of each building block is 2. Each fitness will be between 0 and 17 when $1 < C_{bb} < 2$ since the fitness of an individual in $H_1$ is $(\Sigma 3 \cdot C_{bb}) \cdot S$. Figure 5 illustrates 3 examples of fitness evaluation with BB1.

![Building Blocks encircled and labeled A,B,C,D,E.](Image)

**Figure 4 BB1**

*Note that for $N>2J$ the discount will be negative but it is no problem to reset the fitness to zero.*
Figure 5 Examples of Fitness Using BB₁

BB₁ is designed to provide a collection of building blocks in which the combination of blocks together does not contribute more to the fitness of a program than the sum of all fitnesses. The conjecture of the GP Building Block Hypothesis is that hierarchical combination of building blocks speeds convergence. So, in contrast to BB₁, in BB₂ (see Figure 6) intermediate building blocks which have fitness in addition to the simple sum of their component fitness are discerned. BB₂ is intended to check whether the combination of fittest compact lowest order schemas into much fitter higher order schemas increases the speed of GP convergence. If the Building Block Hypothesis is correct, GP should converge upon the optimum faster with BB₂ than BB₁. The intermediate level building blocks should act as stepping stones since their presence is explicitly rewarded via additional fitness value. The combination of small building blocks at one level into bigger blocks one level higher is assumed to take place when the subtrees which are small blocks are connected up with one another via an additional node acting as a root. Thus the progressively larger building blocks in BB₂ are the combined size of the components plus one. Note that the longer building block's added value to an individual is the sum of its component building blocks' values; it is a linear contribution. Because the higher level building block
must be assessed for compactness, the 'net value' or effect on fitness when a program contains it is always more (because $1 < C_{bb} < 2$)

Table 1 and Figure 7 show the net effect of the building blocks before the size discount in different programs. Note that a 'stepping stone' in GP is more complex than in a GA: it depends on the value (i.e., order) and compactness of a building block. There is no way the 'stepping stones' can be of constant increment because in GP the size of trees is variable, making the compactness variable.

**Figure 7 Individuals of For BB2 Fitness Examples**

<table>
<thead>
<tr>
<th>Building Block</th>
<th>Value in Perfect Individual</th>
<th>Value in Fig 7 Tree A</th>
<th>Value in Fig 7 Tree B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>5.6</td>
<td>5.54</td>
<td>5.74</td>
</tr>
<tr>
<td>Level 2</td>
<td>11.2</td>
<td>10.76</td>
<td>12.17</td>
</tr>
<tr>
<td>Level 3</td>
<td>16</td>
<td>-</td>
<td>20.87</td>
</tr>
</tbody>
</table>

**Table 1 Net Effect of BB2 Building Blocks in Programs**

**Section 4: Other BB Functions:** BB2 designates each bigger building block as the hierarchical connection of two smaller building blocks using an additional 'connecting root'. This combination is the minimal disruptive manner of combining the smaller blocks because a connecting root is a minimum length path which could join them. But there is also the possibility that the explorative effect of crossover works to rearrange lower order building blocks into suitable combinations once groups of them start to occur in single programs. This phenomenon is seen over the population as exchange of material. The rationale for discounted rewarding of building block combinations which are not joined minimally would be to investigate the issue of whether a sub-layer of building blocks is useful. The sub-level may ensure that a collection of more than one block (linked in individuals by disruptible paths) will bring new and ultimately more successful combinations into existence. This focuses upon the detailed

*This can be changed to always being less by changing the compactness definition to $(N-DL)/N)$ so $0 < C_{bb} < 1$. 

10
behaviour during the second stage of GP search in which low order building blocks combine into longer ones. For this reason, another set of BB functions for GP: BB3, ..., BB6, is suggested in Figures 8 (a) - (d) and 9. These functions investigate identifying building blocks which are of increasing size but which are not always interconnected as one extended subtree. BB3 and BB4 could show that GP not only operates by the progressive extension of partial solutions as suggested in BB2 but it also can collect partial solutions together not by just gradual enlargement but, by what is more aptly described as amassing larger collections of building blocks in sub-layers.

BB3 and BB4 differ by their level 2 building blocks. In BB3 any pairwise combination of level 1 building blocks is identified as a level 2 building block whereas in BB4 only the 2 pairs (A and B or C and D) that must eventually be joined directly together are identified. (The building blocks which

Figure 8 (a) and (b) BB3 Level 1 and 2 Building Blocks

Figure 8 (c) and (d) BB3 Level 3 and 4 Building Blocks
identify the combination are level 4, they are the subtrees rooted at E and F.) A comparison of BB3 and BB4 would indicate how useful it is to have information about combinations which may not link directly together. Also, it could illustrate different exploration and combination behaviour of the crossover operator. BB3 creates a 'more fertile' platform for finding the level 4 building blocks but does the crossover operator have the power to find one? It could be the case that too many incorrect combinations are found. In this case the level 3 building block (which is the presence of all 4 level 1 building blocks) is also counted.

Pairs of trees which will eventually be joined into one larger tree (A,B) and (C,D). Note that levels 1,3,4,5 building blocks are the same as BB3.

Figure 9 BB4 Level 2 Building Blocks

BB5 and BB6 (not shown) act as respective counterparts to BB3 and BB4 except that they ignore the level 3 building block. Compared to BB3 and BB4 they therefore increase the relative importance of their respective level 2 building blocks. The current theory does not indicate whether this emphasis is significant.

Are level 3 building blocks valuable? This can be answered by comparing BB4 to BB5 and BB3 to BB5. The possibility exists that level 3 building blocks may be irrelevant or even misleading.

BB7 Building Blocks within a level have diverse values. Values are noted beside the root. (E.G. Subtree E has value 13)

Figure 10 BB7 Building Blocks
As opposed to the Hierarchy series which assigns equal values to building blocks of the same order, instead the values are varied in the GP Diverse Building Blocks functions. The scenario is that the building blocks are above average in fitness but that they differ in how much better they are. This is a valid event in programs. Solving a particular subproblem may provide a more crucial step or general purpose step and this may be reflected in the fitness function by the composition of the test suite and weighting of test cases. However, eventually it is the synergy of all components that counts. All of BB7,..., BB9 use the building blocks identified in BB2; Figures 10-12. In BB7 all blocks in level 1 have different values but subsequent higher level blocks are a linear combination. Various weightings of level 1 blocks can be tried. In BB8 each level 1 block of a pair which combine at the next level is given the same value but each pair adds to a different value. At the next level the values are the same. In BB9 the values at the next level counter-balance the weights of level 1. BB8 and BB9 experiments can be used to study the presence of epistasis in programs and are more complicated than BB7 because they do not simply investigate diversity in one layer but they investigate non-linear values amongst layers.

**Figure 11 BB8 Building Blocks**

BB8 building blocks within a level have diverse values BUT at a higher level are of equal value. Values are noted beside the root. (E.G., Subtree E has value 9)

**Figure 12 BB9 Building Blocks**

BB9 building blocks within a level have diverse values BUT at a higher level are counter balanced. Values are noted beside the root. (E.G., Subtree E has value 4)
Space does not allow us to describe more potentially insightful BB functions. See [O'Reilly 1994] which considers how to examine the role of "junk genes", evaluate the effects of different population initialization methods, how to compare crossover operators and how to characterize circumstances when non-concurrent detection of equally valued building blocks is possible.

**Conclusion:** One explanation for GP's success when it occurs is that GP has discovered building blocks, their numbers have increased and they have been combined into increasingly larger sub-solutions (and eventually the solution). We have presented BB functions which allow the verification of this conjecture. The design of such functions for GP must account for GP's variable length and hierarchical representation properties. The design is quite straightforward and we feel that experiments thus conducted will be more convincing and illuminating than subprogram ancestry tracing which is currently lightly offered as evidence [Koza 1992]. We note that all BB function experimentation is predicated upon the assumption that building blocks (an intrinsic property of both the primitives and fitness function of a GP problem) do exist. We also advocate investigation of other possible explanations which may not be based upon the presumed existence of building blocks.

**Acknowledgements:** Both authors thank Mark Wineberg. Ms. O'Reilly would also like to thank Stephanie Forrest, members of the U.N.M. study group, Terry Jones, Bill McCready, Melanie Mitchell and members of the unique research environment at the Santa Fe Institute who have been considerably helpful in discussing this work.

**References**


