Building Block Emergence in Genetic Programming

John Aleshunas

Background

Genetic Programming (GP) is an evolutionary method that is adept at solving optimal instruction set problems. Some examples of problems that GP is well suited for are: computer programs, logic circuits, and mathematical equations [**2**]. Genetic programming often represents a population of candidate solutions as trees. These trees are composed of elements from a predetermined function set and terminal set. The members of the function set will have some number of operands which can be satisfied by members of both the function set and the terminal set. New solution populations are evolved from these population members using genetic operators such as crossover - where sub-trees are exchanged between two parent solutions - mutation - where a sub-tree of a solution is randomly modified to create a new candidate solution – and selection – where fit population members are chosen to participate in the generation of the next population of solutions.

One critical issue with GP design is the selection of the members of the function and terminal sets. These sets should be complete enough to be able to adequately represent a correct solution for the problem. Unfortunately, if these sets include too many elements, the search of this representation space becomes so large that it impacts the performance of the GP [**2**]. Methods that can help discover the best subset of functions and terminals thereby reducing this representation space will improve the performance and solution quality of a GP implementation.

A starting point in developing an efficient method to reduce the GP representation space is the understanding of why an evolutionary process, such as a GP, actually works. Early efforts by John Holland to explain why genetic algorithms worked resulted in his well known schema theory [**7**]. In Holland's schema model, schemata are similarity templates representing entire groups of chromosomes. The schema theorem describes how schemata are expected to propagate generation after generation under the effects of selection, crossover and mutation operators. More recently, genetic algorithm theorists have explored the use of Markov chain models to develop theoretical results on the convergence properties of GAs [**3**]. While the results of Markov chain models are important in principle, their practical use is limited because they lead to computationally intractable equations even for tiny populations of very short bit strings. Schema theorems, on the other hand, lead to relatively concise descriptions of the way GAs search and are easily understood [**13**]. It is therefore natural to extend this concept to genetic programming and define schema for GP trees.

This paper will review several schema or building block concepts applied to genetic programming, discuss how these concepts explain the convergence of GPs, highlight what the implications of these concepts have on GP behavior, provide some empirical results demonstrating these implications and demonstrate how this process can be improved by encouraging the emergence of desirable building blocks.

Schema as Program Components or Building Blocks

Schema theorems are probabilistic models that attempt to explain why evolutionary processes work. They model how the number of fit schema in a population of solutions will grow based on the effects of selection, mutation and crossover. This section will review general schema theory, several versions of schema theory for genetic programming and their implications for GP behavior. While other representation structures exist, this discussion will concentrate on standard linear string-based genetic algorithms and tree-based GP implementations since most of the schema theorems discussed here are based on these representation models.

John Holland conducted early research in this area to explain the behavior of genetic algorithms [**7**]. He defined schema as similarity templates representing entire groups of chromosomes. In his work, Holland developed his schema theorem which predicts how the number of strings in a population matching a schema is expected to vary from one generation to the next under the effects of selection, bit-flip mutation, and one-point crossover. This theorem is mathematically stated as:

$$E[m(H,t+1)] \geq m(H,t) \cdot \frac{f(H,t)}{\bar{f}(t)} \cdot (1-p_m)^{O(H)} \cdot \left[1 - p_{xo} \frac{L(H)}{N-1}\left(1 - \frac{m(H,t)f(H,t)}{M\bar{f}(t)}\right)\right] \qquad (1)$$

The first component on the right side of equation (1) identifies the number of strings matching the schema H at generation t. The second component accounts for the effect of selection by computing the ratio of the fitness of the schema H versus the mean population fitness. This term implies that fitter schema will have a higher probability of selection from generation to generation. The third component accounts for the effects of mutation. It explains how mutation impacts the growth of the number of a schema and is affected by the order – or defining length – of the schema H. The final component accounts for the effects of crossover. This component describes both the constructive and destructive effects of crossover both of which are affected by the defining length [*L(H)*] of the schema H and the number of strings matching H at time t [*m(H, t)*].

Goldberg extended this model into his building block hypothesis by stating that GAs work by combining relatively fit, short schema to form complete solutions [**4**]. This hypothesis, along with Holland's schema theorem, assert that (a) fitter schema are selected for operations at a higher probability than less fit schema, (b) crossover mixes and concatenates low order schema components to form higher order ones (c) mutation has less of an impact on smaller fit schema and (d) the positive effects of these operators on the population outweigh the negative effects.

John Koza attempted to explain why genetic programming works by extending Holland's schema model and theorem to GP [**12**]. He represented schema as a set of program subtrees or S-expressions. This definition of schema was probably suggested by the fact that Koza's GP crossover moves subtrees between parents and children. One significant difference between Koza's schema definition and the conventional GA schema is that Koza's definition gives only the defining components of a schema and not their position. The same GP schema can potentially be instantiated in different ways in the same program. This is how his model accounts for two key differences between GA representations and GP representations: (1) GA representations are often fixed size whereas GP representations and usually variable sized and (2) GA normally have a semantic interpretation for a given location in a chromosome while GP solutions often do not exhibit location dependent meanings.

Altenberg produced what can be considered the first mathematical formulation of a schema theorem for GP [**1**]. This theorem simplifies some of the GP operating environment by assuming that the population is very large, there is no mutation, and that fitness proportionate selection is used. Given these assumptions, this model does describe how the effects of selection and crossover are affected by the ratio of a schema's fitness to the mean population fitness. This theorem is mathematically stated as:

$$\frac{m(i,t+1)}{M} = (1-p_{xo})\frac{f(i)}{\bar{f}(t)}\frac{m(i,t)}{M} + p_{xo}\sum_{j,k\in P}\frac{f(j)f(k)}{\bar{f}^2(t)}\frac{m(j,t)}{M}\frac{m(k,t)}{M}\sum_{s\in S}P(i \leftarrow j,s)C(s \leftarrow k) \qquad (2)$$

His equation (2) contains components for selection and crossover the effect of each is adjusted by the fitness of the schema of interest relative to the population fitness and it predicts the expected number of instances of the schema in the next generation. Altenberg's concept of a schema differs from that of Koza's. Koza defined a schema that could be made up of multiple subexpressions. Altenberg defined his schema as a single subexpression.

O'Reilly and Oppacher derived a schema theorem for GP with fitness proportionate selection and crossover formalizing and refining Koza's work on schemata [**17**]. They defined a schema as an unordered collection of subtrees and tree fragments. This definition gives only the defining components of a schema, not their position. Such a schema can be instantiated by a program in different ways, and therefore multiple times. This concept is again demonstrates the difference between GA schema – where location in a chromosome implies meaning – and GP schema – where location in a tree is not as important as the overall evaluation of the solution. This schema definition allowed O’Reilly and Oppacher to define the concepts of order and defining length for GP schema. While flexible, this definition is complicated by the fact that the components for a schema can be embedded in different ways in different programs. Additionally, they discovered that the accuracy 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 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 [**17**]. This theorem is mathematically stated as:

$$E[i(H,t+1)] \geq i(H,t) \cdot \frac{f(H,t)}{\bar{f}(t)} \cdot \left(1 - p_{xo} \cdot \max_{h \in Pop(t)} P_d(H,h,t)\right) \qquad (3)$$

Besides including component terms for fitness proportional selection and crossover, equation (3) also includes a term that accounts for the probability of disruption $P_d(H,h,t)$ of a schema *H* contained in the program *h* at generation *t* due to crossover.

One outcome of O’Reilly and Oppacher’s work is the development of a definition of a GP building block and a GP building block hypothesis. They define a GP building block as a low order, consistently compact GP schema with consistently above average observed performance that is expected to be sampled at increasing or exponential rates in future generations. This essentially translates Goldberg’s GA building block hypothesis into a GP building block hypothesis which states that GP combines building blocks, the low order, compact highly fit partial solutions of past samplings, to compose individuals which, improve in fitness. Thus, the source of GP’s power, when it works, lies in the fact that selection and crossover guide GP toward the evolution of improved solutions by discovering, promoting and combining building blocks.

Recent work by Sastry, O’Reilly, Goldberg and Hill has further explored the behavior of building blocks and their expression in GP [**20**]. They defined GP schema or building blocks as single tree patterns with each of its nodes labeled with a single function or terminal symbol. Their building blocks have no absolute position or positional anchor and therefore can be instantiated anywhere in a program. Since each building block is not anchored to a position of a tree, there can be none or more than one instance in a single tree. This conceptualization of GP building blocks highlights that while counting number of instances of tree fragments can itself be important, what is more important are those tree fragments that get expressed. The expression mechanism in GP dictates what the building blocks of a problem are and therefore affects the building block supply. Therefore the expression of small tree fragments into partially correct subfunctions is important to GP success. Using this building block concept they developed a model to estimate the population size required to ensure the presence of at least one copy of all raw building blocks in the initial population. This population sizing model also indicates that there is a minimum tree size dependent on the problem size. These two conclusions, that for a given problem the initial building block supply is dependent on tree size and population size, are potentially useful in tuning GP solution convergence.

Context-Free Grammar – GP (CFG-GP) is a method to adapt the basic GP model so that the CFG-GP can discover the fittest solution structures. In this GP model, the individuals in the population are derivation trees which can be converted into the corresponding programs and the search proceeds using special crossover and mutation operators which always produce valid derivation trees. Whigham produced a definition of schema for context-free grammars and the related schema theorem [**23**]. He defines a schema as a partial derivation tree rooted in some non-terminal node. For a CFG-GP this means that a schema represents all programs that can be obtained by completing the schema. Whigham’s definition leads to equations that predict the probability of disruption of schemata under crossover and mutation. Similar to O’Reilly’s GP schema theorem, these probabilities vary with the size of the tree matching the schema.

While all of these different schema theorems develop different models that attempt to explain why evolutionary processes work, they all share some key aspect and issues. The next section explores these general aspects of GP operation and discusses the limitations of the schema theorems in fully explaining GP results.

Generalizations and implications from GP schema theory

The previous section highlighted several GP schema theorems and explained what assertions they made regarding how a GP develops a solution. The key common concept is that fit schema increase in number from generation to generation thereby creating fitter solutions in the population. This is accomplished using the operators of mutation, crossover and selection.

Mutation acts to retain diversity in the population by randomly altering a selected subtree in a population member that is chosen for mutation [**2**]. In a simplified form, mutation can be thought of as a random search operator. Less fit building blocks can be modified into fitter building blocks by mutation. Additionally, mutation can disrupt fit building blocks. The primary determinants of how much impact the mutation operator will have on a population of fit building blocks are the mutation probability – which is usually small, the size of the building blocks – smaller blocks are affected less, and the size of the fit building block population – a small mutation probability has less of an effect on a large population of fit building blocks.

Crossover is the primary mixing operator in GP [**2**]. Like mutation, crossover can both improve a building block and disrupt it. It has a greater impact –both positive and negative - on larger building blocks much like mutation. Another aspect of crossover is its tendency to grow larger trees over several generations. This behavior becomes a limiting aspect in GP as a population evolves. This tendency to grow larger trees is called bloat and it can stifle the ultimate fitness of a population of solutions. Crossover, like mutation, is a random search operator.

Selection is the engine that causes a GP to converge to a solution [**2**]. The concept of the building block hypothesis is based on how fitness selection chooses fit individuals – consisting of fit building blocks – to be subject to crossover and mutation. This implies that fit components are mixed or mutated to form fitter individuals in the population from generation to generation and is the basis for the building block hypothesis.

All of the schema theorems discussed above computationally describe how these operators contribute to the functioning of a GP. There are some caveats to these models. First it should b e noted that GP schema theories often involve assumptions that simplify the expectation probability equations. These assumptions include using mutation, restriction to fitness proportional selection or restricting the crossover operator to one-point crossover [**17**]. Nevertheless, the general GP behavior is still accounted for.

Another limitation of most schema theorems is that they describe the change in the number of a particular schema in the population – given a set of operator parameters – from one generation to the next. They do not predict schema behavior over a span of many generations. It is assumed that if a schema will increase or decrease for one generation to another, then this behavior will continue over a span of generations. Unfortunately, since GP schema theorems do not describe behavior for more than one time step and it is not the case that the required behavior is constant [**17**].

The selection mechanism in many schema theorems is the ratio of the desired schema's fitness relative to the average population fitness. This concept, while mathematically concise, hides two issues. First, the fitness of building blocks normally cannot be evaluated separate from the individual population member is part. Most solutions and their fitness functions are not decomposable. This means that the fitness ratio that supports the selection component in any given schema theorem normally cannot actually be evaluated. Secondly, if we assume we are going to use this fitness ratio to describe how schemas that are fitter than the population average will be preferred over other less fit schemas, we are left with a diminishing selection force. As fit schemas take over the population, the average fitness of the population rises while the fitness of the schemas does not change. This means that as a schema starts dominating, the margin by which it is fitter than the average fitness of the population decreases and the selection pressure decreases [**17**].

One final issue inhibits GP solution development. The issue is bloat, where GP tends to evolve individuals that grow uncontrollably until they reach the maximum tree depth and the GP fitness plateaus. A body of research has established that GP bloat is caused by introns [**2**]. Introns are subtrees within our solutions that do not contribute to the overall fitness of the solution. A subtree in a solution to a regression problem that evaluates to 1 and is multiplied in its connection to the overall solution is an example of an intron. This surplus code consumes space in the solution tree but contributes nothing to the fitness. Once a tree reaches its maximal size the effects of mutation and crossover become muted. The issues associated with bloat and introns are closely related to the tree size and the initial building block supply. This is often the reason the fitness of a given GP will eventually plateau and not improve.

The following section will present the results of several experiments constructed to demonstrate the behaviors of GP building blocks.

Emergence of building blocks – empirical results

Schema theorems provide a mathematical model that describes how GP work. The primary mechanism described is the instantiation of these theoretical schemas into instantiations as actual building blocks in the GP population [**22**]. This section discusses several experiments that attempt to see if the building block behavior described in these models actually occurs. The focus of this discussion will be compact, fully defined building blocks with no wild-card nodes that follow two pattern models: 1st order and 2nd order building blocks. 1st order building blocks are GP subtrees consisting of a function node and one of its child nodes. The equation $(x * x)$ consists of two 1st order building blocks: multiply with first argument x $\{*_1, x\}$ and multiply with second argument x $\{*_2, x\}$. 2nd order building blocks are GP subtrees consisting of a function node and all of its child nodes. The equation $(x * x)$ consists of one 2nd order building block: multiply with an x as both arguments $\{*, x, x\}$. All of the experiments in this paper attempt to solve a regression problem described by the Bowl3 equation

$$(x * x) + (y * y) + (z * z) \qquad (4)$$

The Bowl3 equation (4) is useful for these experiments because the desirable 1st order and 2nd order building blocks are easily determined and tracked. This equation has ten desirable 1st order building blocks – subscript indicates the function argument number.

$$\{+_1, +\}, \{+_2, +\}, \{+_1, *\}, \{+_2, *\},$$
$$\{*_1, x\}, \{*_1, y\}, \{*_1, z\}, \{*_2, x\}, \{*_2, y\}, \{*_2, z\}$$

These 1st order building blocks can be combined to form thirteen 2nd order building blocks

$$\{+, +, +\}, \{+, +, *\}, \{+, *, +\}, \{+, *, *\},$$
$$\{*, x, x\}, \{*, x, y\}, \{*, x, z\},$$
$$\{*, y, x\}, \{*, y, y\}, \{*, y, z\},$$
$$\{*, z, x\}, \{*, z, y\}, \{*, z, z\}$$

Not all of these building blocks are used to solve the problem. Equation (4) has six desirable 2nd order building blocks in this set.

$$\{+, +, *\}, \{+, *, +\}, \{+, *, *\},$$
$$\{*, x, x\}, \{*, y, y\}, \{*, z, z\}$$

The other seven building blocks do not contribute to the solution. In these experiments the function set consisted of four functions – addition, subtraction, multiplication and protected division. The terminal set consisted of fourteen elements – x, y, z, and the constants -5 to 5. These functions and terminals extend to 162 potential 1st order building blocks and 2592 potential 2nd order building blocks.

The initial experiments used a modified form of Lil-gp 1.02 [**24**] developed by Cezary Janikow [**8**]. This GP application has the ability to track the individual building block frequencies in each generation. In these experiments the application was run in standard GP mode using mutation, crossover and tournament selection. Each experiment consisted of thirty independent runs using different random seeds running for 250 generations. The building block frequencies are measured using a subset of 10% fittest trees for each population size. The initial experiment explored how building block supply was related to population size [**20**].

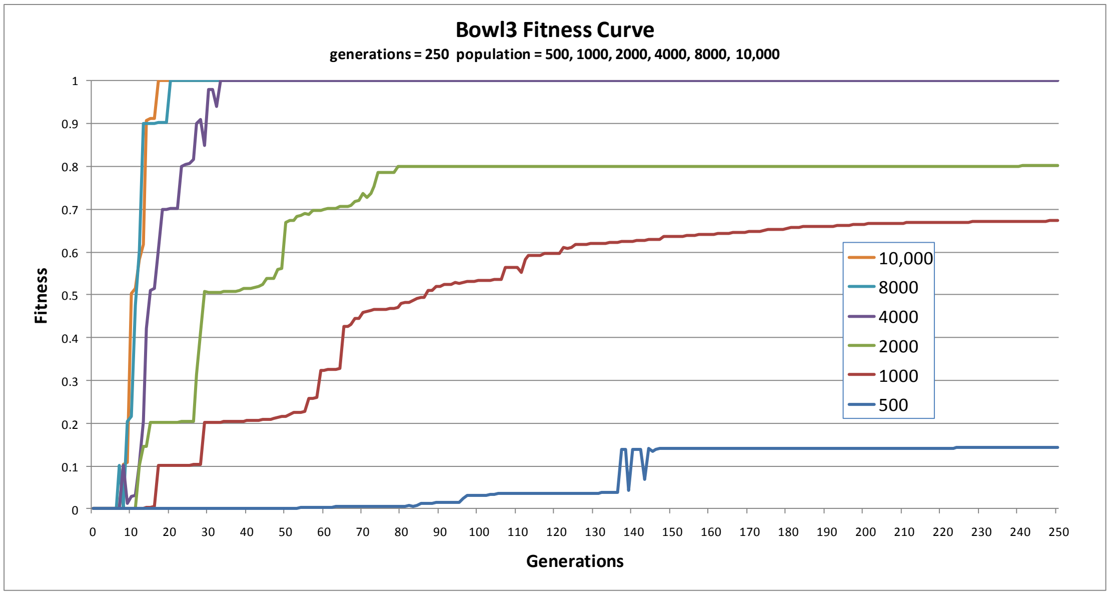


Figure 1 Fitness Curve for the Bowl3 Equation for Various Population Sizes

Figure 1 shows the average fitness for six sets of GP experiments different population sizes. The largest populations (4000, 8000, and 10,000) quickly find a good solution. The middle sized populations (1000 and 2000) achieve good fitness but appear to plateau at a best achievable fitness value. The smallest population does not evolve a good solution. Additionally, it appears to exhibit an abrupt fitness change near generation 140.

The easy convergence of the large populations may clearly be attributed to the rich supply of building blocks [**5**], [**6**], [**20**]. This large supply of fit components should mean that there are many fit solutions available in early generations that contribute to the early convergence and high solution fitness. The middle sized populations appear to develop an adequate supply of fit building blocks but these populations also seem to develop bloat thereby inhibiting the overall quality of the population fitness. The smallest population never evolves a strong solution. The population size probably restricts the initial supply of fit building blocks and it takes so long for desirable building blocks to emerge that all progress is halted by GP bloat [**2**].

While these explanations follow from the schema theorems discussed in this paper, an examination of the actual building block counts, from generation to generation, should confirm or refute these assertions. The following figures portray each population's count of specific building as a percentage of all building blocks in the fittest 10% of the trees in each generation. This metric allows for easy comparison of each experiment independent of the population size or sample size.

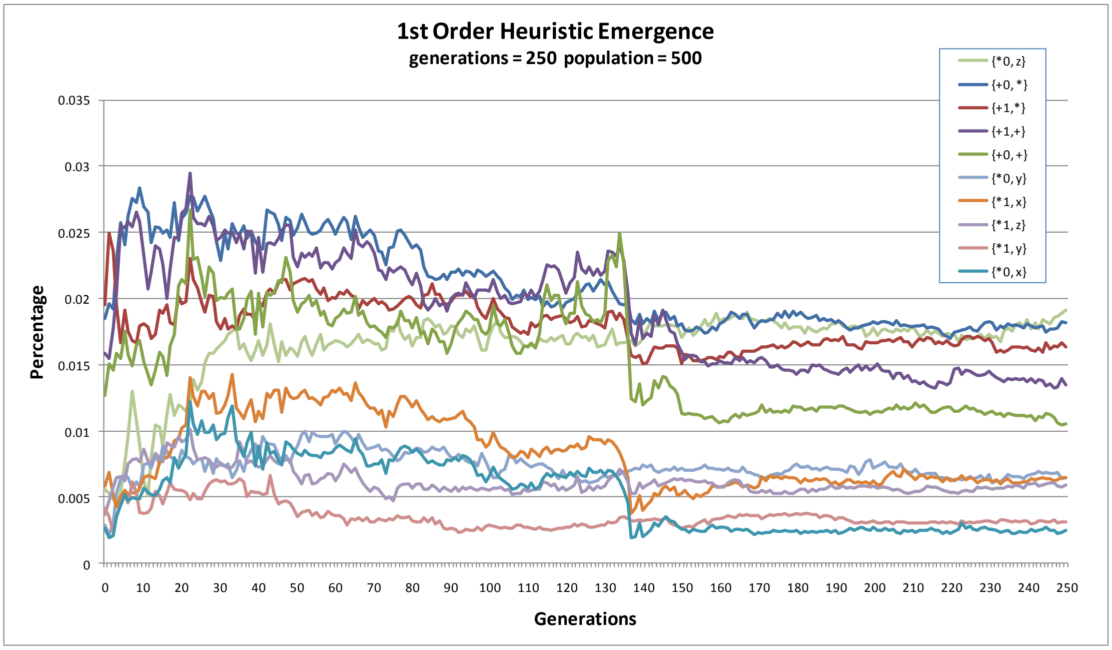


Figure 2 1st Order Building Blocks for Bowl3

The ten desirable 1st order building blocks of the Bowl3 equation are shown in Figure 2 for a population of 500 individuals. The sharp drop near generation 140 corresponds to the rise in fitness in Figure 1. While it is clear which building blocks are ignored, it is not completely clear from these building blocks why they are ignored and why this results in a jump in the fitness. Examination of the 2nd order building block frequencies may give us more definitive information.

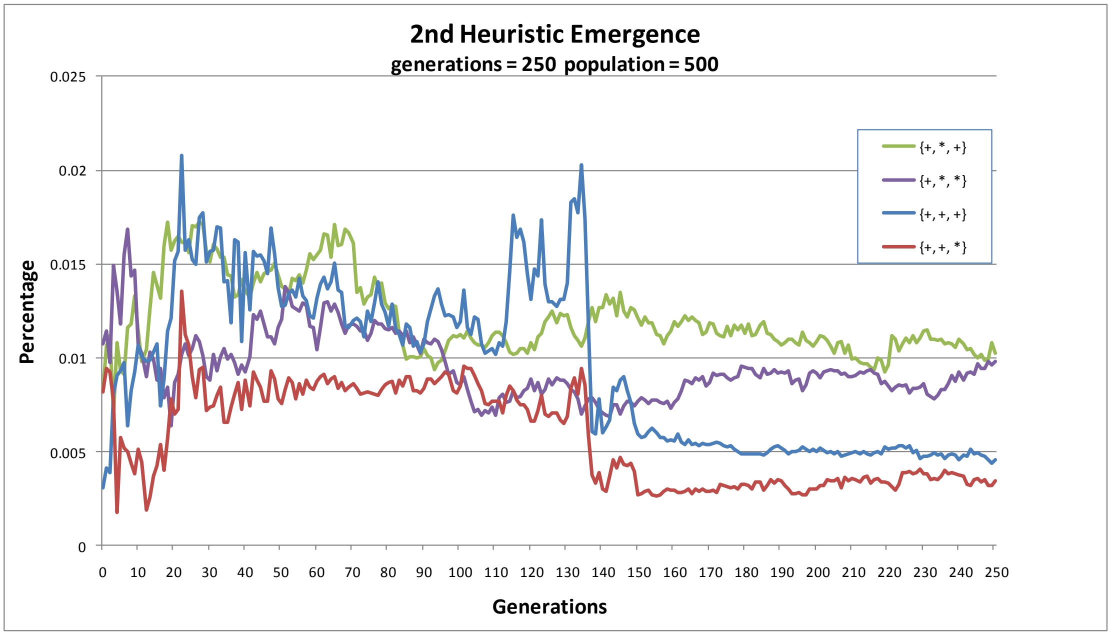


Figure 3 2nd Order Building Blocks for Bowl3 (addition root)

Figure 3 shows the 2nd order building blocks for Bowl3 where addition is the root of the subtree. This figure clarifies the source of the fitness jump. The building block {+, +, +} is frequent in the population in the early generations. This building block is not necessary to solve Bowl3. Once this building block is ignored near generation 140 the other more desirable building blocks increase their influence on the GP fitness. The other building block that experiences a drop near generation 140 in Figure 3 is $\{+,+,*\}$. While this is a desirable building block, it is the reflection of another desirable building block $\{+,*,+\}$ and the suppression of this building block helps stabilize the solution structure.

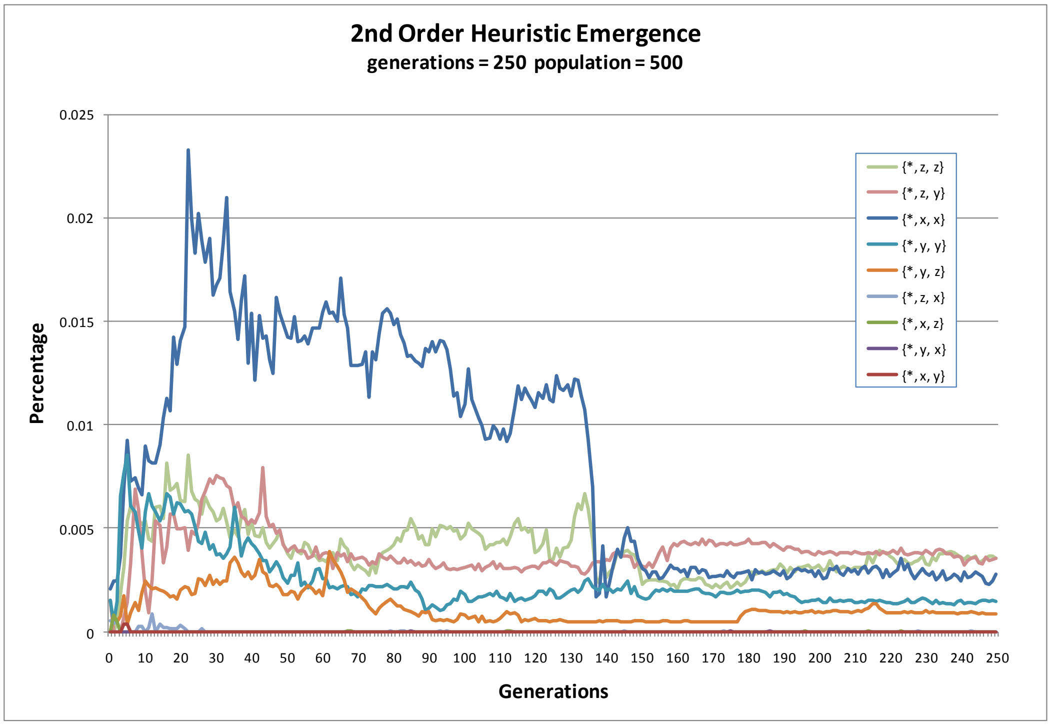


Figure 4 2nd Order Building Blocks for Bowl3 (multiplication root)

Figure 4 shows the 2^{nd} order building blocks for Bowl3 where multiplication is the root of the subtree. While it is clear that the building block $\{*,x,x\}$ has a strong early emergence, the other desirable 2^{nd} order building blocks $\{*,y,y\}$ and $\{*,z,z\}$ emerge more slowly. The shift in the population's building blocks that takes place near generation 140 is also evident. Most of the other 2^{nd} order building blocks are ignored from the start of the GP execution.

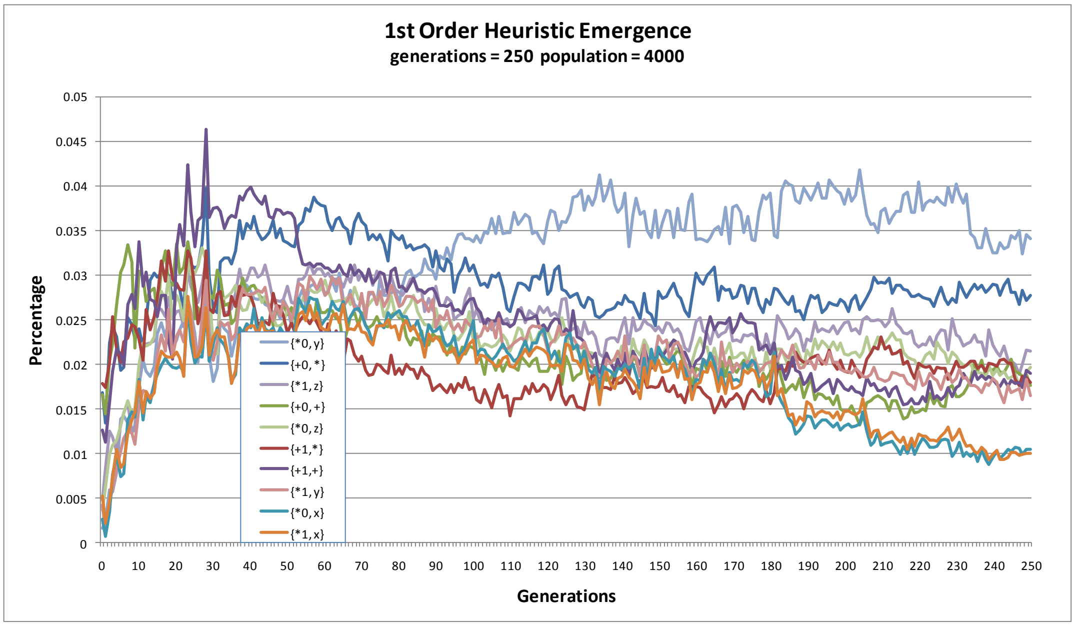


Figure 5 1st Order Building Blocks for Bowl3

Figure 5 shows the growth of 1^{st} order building blocks for a population of 4000. This chart clearly shows that the building block supply is rich enough to quickly develop fit solutions with a population this large. Another behavior worth noting in this figure – and all of the other building block frequency figures – is how the frequency percentiles of the specific building blocks fluctuate from generation to generation. It was previously pointed out in this paper that while schema theorems estimate the number of instances of a given schema from one generation to another they do not make any assertions regarding the schema instance growth over a span of generations. Figure 5 shows that the growth in the frequency of building blocks is not smooth. These fluctuations are in fact illustrative of the constructive and destructive effects of the GP operators from generation to generation.

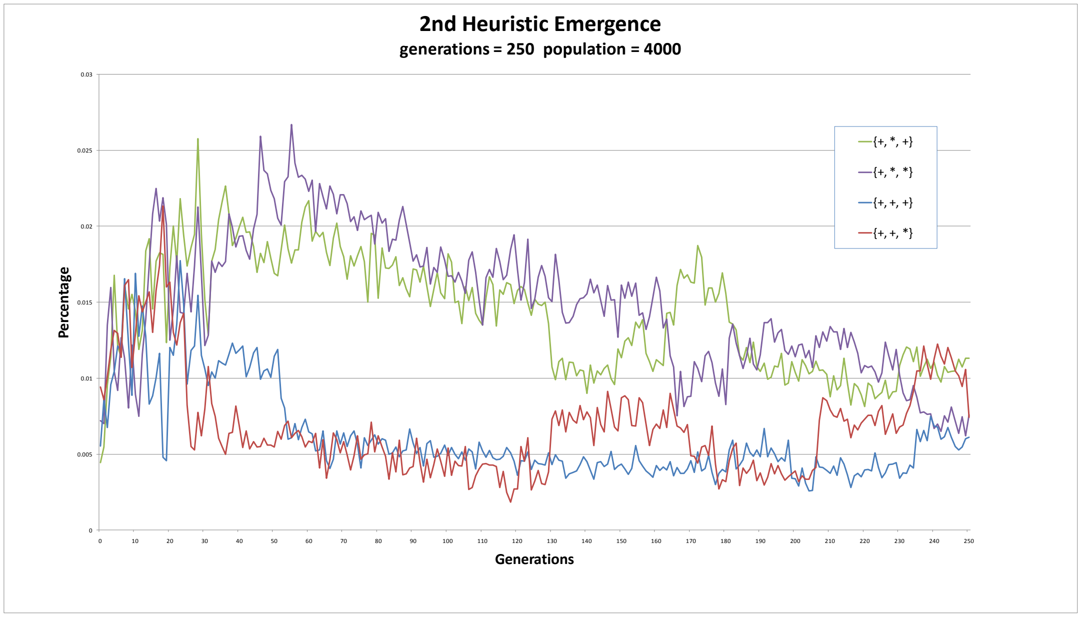


Figure 6 2nd Order Building Blocks for Bowl3 (addition root)

Figure 6 shows the 2nd order building blocks for Bowl3 where addition is the root of the subtree for a population of 4000. Again it is clear that the GP has a good early supply of fit building blocks to evolve quality solutions. The undesirable building block $\{+,+,+\}$ is quickly ignored because the other desired building blocks are readily available in the initial population.

Comparing the percentage density of the building blocks as the population size increases shows how bloat can impair the development of quality solutions in a GP if desirable building blocks are not expressed in early generations.

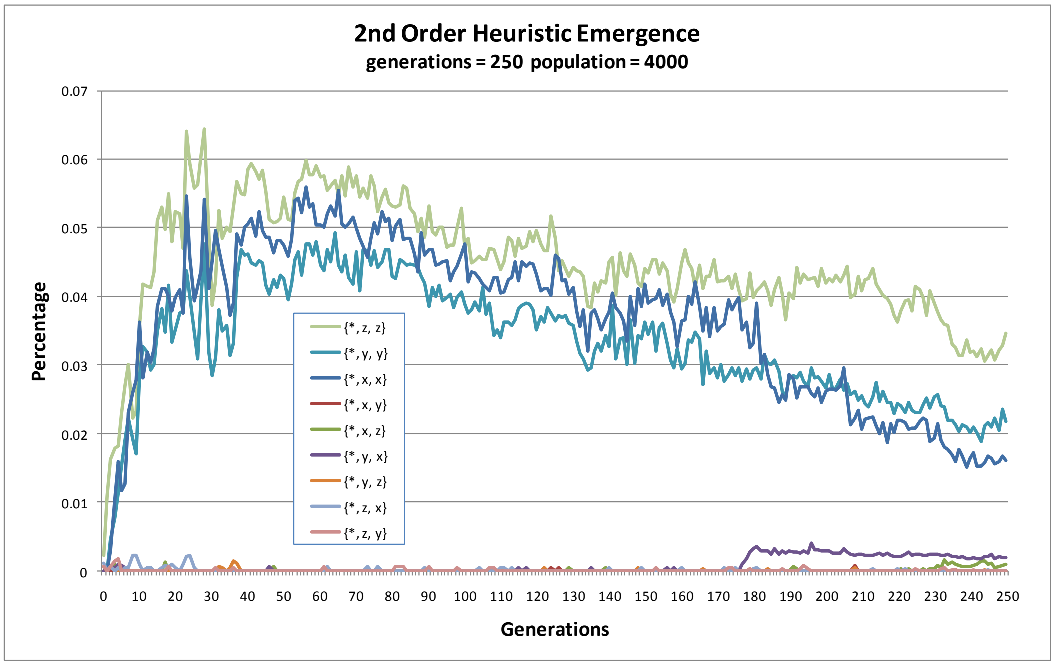


Figure 7 2nd Order Building Blocks for Bowl3 (Multiplication root)

Figure 7 shows the 2nd order building blocks for Bowl3 where multiplication is the root of the subtree for a population of 4000. The population in this experiment is large enough that the three desirable 2nd order building blocks with multiplication as their root $\{*,x,x\}$, $\{*,y,y\}$ and $\{*,z,z\}$ are dominate in the population and all other building blocks are ignored. This result demonstrates how a rich population of available building blocks enables a GP to formulate quality solutions.

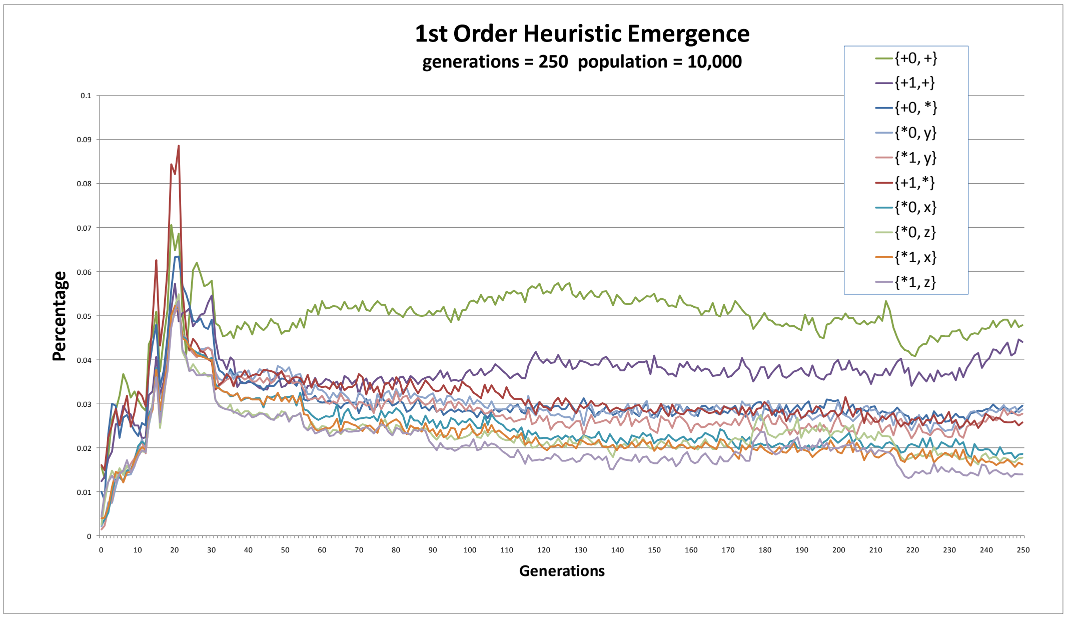


Figure 8 1st Order Building Blocks for Bowl3

Figure 8 plainly shows the impact of a large 10,000 population on building block supply. The desirable building blocks are readily available in the initial population and the GP quickly converges to a stable fit solution. Furthermore, if the building block percentages are compared between this figure and the figures for the smaller populations, the effects of bloat can be seen. The smaller populations can only develop a lower frequency of desired building blocks because some of these building blocks are not expressed early enough before bloat consumes large portions of the solution trees. The large initial population in Figure 8 guarantees that there is a good supply of desirable building blocks well before bloat sets in. Unfortunately, it does not prevent the initiation of bloat which can be inferred by the still small percentages of these building blocks.

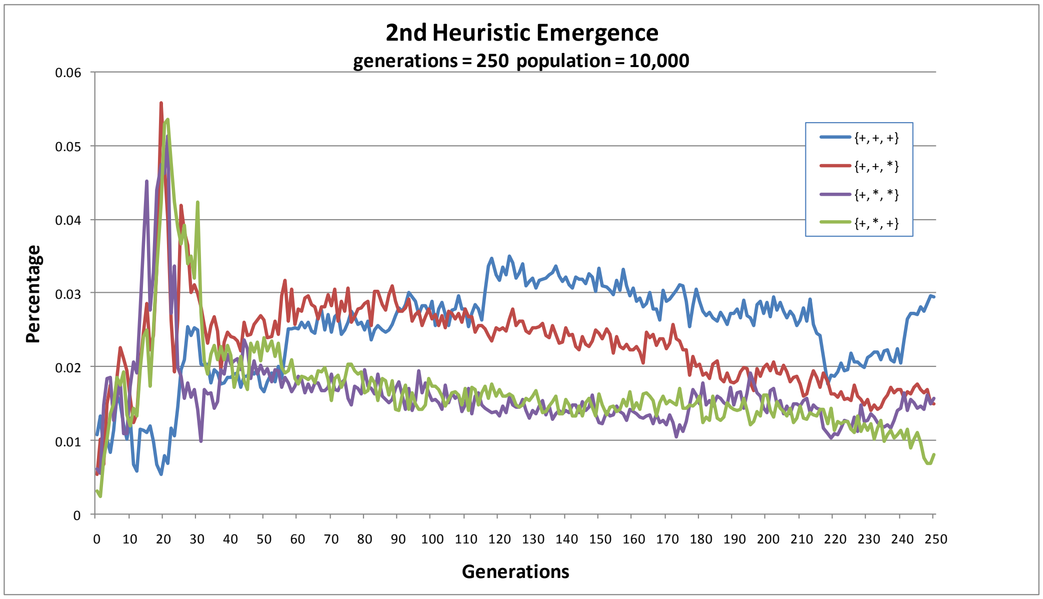


Figure 9 2nd Order Building Blocks for Bowl3 (addition root)

Figure 9 shows the 2nd order building blocks for Bowl3 where addition is the root of the subtree for a population of 10,000. Comparing Figure 9 with Figure 3 and Figure 6 it is apparent that the larger population and its associated good building block supply helps the GP find a good early solution.

Moreover, the building block frequency is markedly higher in the larger population, implying a lower impact from bloat in the final solutions.

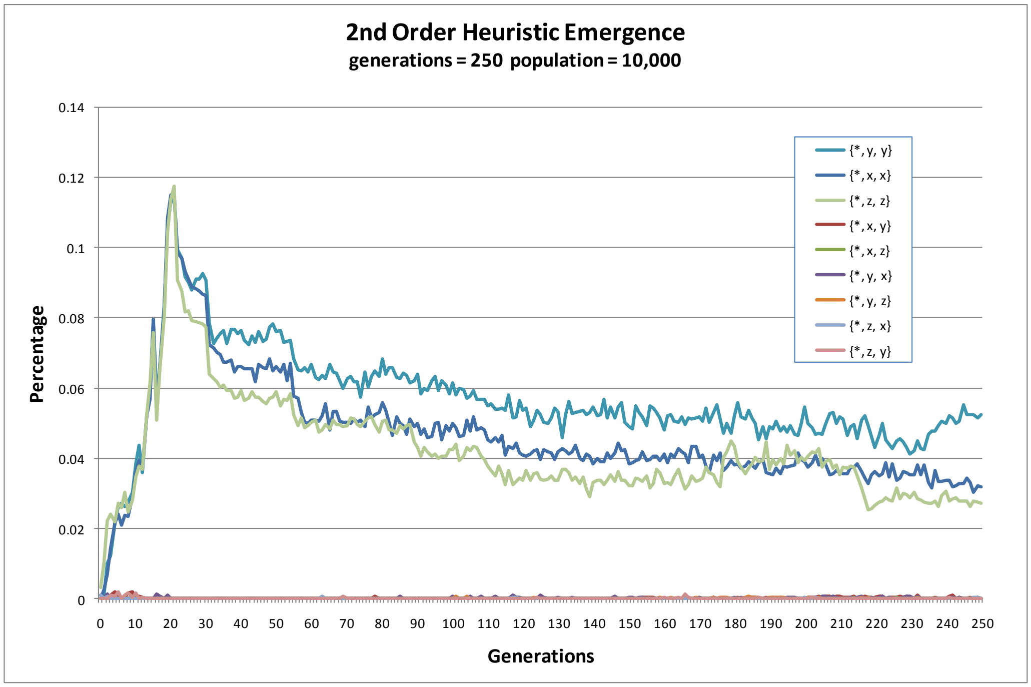


Figure 10 2nd Order Building Blocks for Bowl3 (multiplication root)

Figure 10 shows the 2nd order building blocks for Bowl3 where multiplication is the root of the subtree for a population of 10,000. With a population this large, the desirable 2nd order building blocks are quickly discovered and all other candidates are strongly discouraged. This behavior explains the strong convergence shown in Figure 1 for the GP at very large population levels.

Thus far examination of the observed frequency of the desirable building blocks has illustrated several assertions from GP schema theory: a GP will discover fit building blocks and encourage their further propagation, as fit building blocks emerge they increase population fitness, as fitter building blocks emerge less fit building blocks are discouraged, the change in frequency of building blocks from generation to generation is affected by both the constructive and the destructive effects of GP operators, and a large population of solutions provides a better pool of desirable building blocks. While the frequency of building blocks versus the solution fitness gives some insight on the issue of bloat, different analysis is needed to explore this issue directly.

Table 1 GP Performance by Population Size

Population Size	Execution Time (seconds)	Tree Size (nodes)	Tree Depth (levels)
500	148.7	534.2	19.4
1000	262.2	568.8	19.8
2000	564.1	605.6	19.7
4000	774.4	325.8	18.1
8000	1628.1	454.6	19.0
10,000	2180.3	371.2	18.2

Table 1 shows the average execution time, tree size and tree depth for 30 independent runs of each population size. As expected, the execution time grows with the increased cost of population fitness evaluation. Comparisons within the other two metrics are not as obvious. There is a slight advantage when using larger populations but this does not seem very strong. The lack of a large advantage in populations that have a strong initial building block supply is an implication of the continued negative contribution of tree bloat.

While these experiments demonstrate that a larger population provides a rich pool of building blocks which enables a GP to evolve quality solutions. Furthermore, as desired building blocks emerge their higher fitness improves the fitness of solutions in the population. These fitter solutions are chosen as operation candidates in future generations and this increases the frequency of the desirable building blocks. The frequency of building blocks is subject to the constructive and the destructive effects of the GP operators, but in the larger generational view, fitter building blocks are encouraged and less fit building blocks are discouraged.

There are three remaining issues that limit GP performance improvement. First, how can desirable building blocks be discovered quicker and possibly limit the negative effects of bloat? Next, can this be accomplished with a smaller population thereby reducing the cost of population fitness evaluation? Finally, is there a method that can limit the amount of bloat in population individuals thus increasing the overall solution quality? The next section will present one method that addresses each of these issues and show how its handling of building blocks fosters a potentially efficient evolutionary environment.

ACGP – a method to mitigate negative GP issues

While genetic programming can find novel quality solutions to many programming problems some issues remain limiting the performance of solution evolution. The previous section addressed three significant issues: the timely discovery of desirable building blocks, efficient building block discovery in smaller populations of solutions, and control of tree bloat. Probability-based models similar to GA estimation of distribution algorithms (EDA) [**14**], [**18**], [**19**], [**21**], and Context-Free Grammar GP (CFG-GP) [**14**], [**15**], [**16**] are some methods that have attempted to address these issues. Another method, which will be discussed below, is Adaptable Constrained Genetic Programming (ACGP) [**9**]. ACGP modifies the selection probability of elements in the function and terminal set to encourage desirable building blocks and introduces a new operator REGROW that inhibits bloat formation.

A standard GP treats crossover and mutation as random search operators because during these operations it selects candidates from the set of functions and terminals in a uniform random manner. Selection is this only operator in a standard GP that induces a search bias. Occasionally this scheme may be altered slightly to ensure that only valid trees are generated by the operators.

ACGP modifies this model to include a process that discovers which building blocks potentially contribute to solution fitness and adjusts their selection probability thereby encouraging their propagation in future generations. This process is accomplished by analyzing the frequency of 1st order and 2nd order building blocks in the fittest solutions in all locations in the population trees. Those building blocks that appear frequently in these solutions are assumed to contribute to the solution fitness and should therefore be encouraged. The selection probabilities for these building blocks, and their components, are modified upward. The remaining building blocks are modified downward to ensure that the sum of probabilities equals unity. This analysis and weight adjustment is conducted at intervals over the evolutionary generations. These interval weight adjustments change crossover and mutation into more directed heuristic rather than random searches.

After each weight modification interval, the REGROW operator discards the current population and builds a new population for the next generational interval using the modified weights. This Regrow process accomplishes two things: it creates a fresh population biased toward the desirable building block structures and it flushes bloat out of the population of solutions.

To determine whether these heuristic modifications address the negative issues identified above, the experiments were repeated using ACGP and a small population of 500 solutions. The total number of generations remained 250 with the ACGP analysis intervals set at every 20 generations.

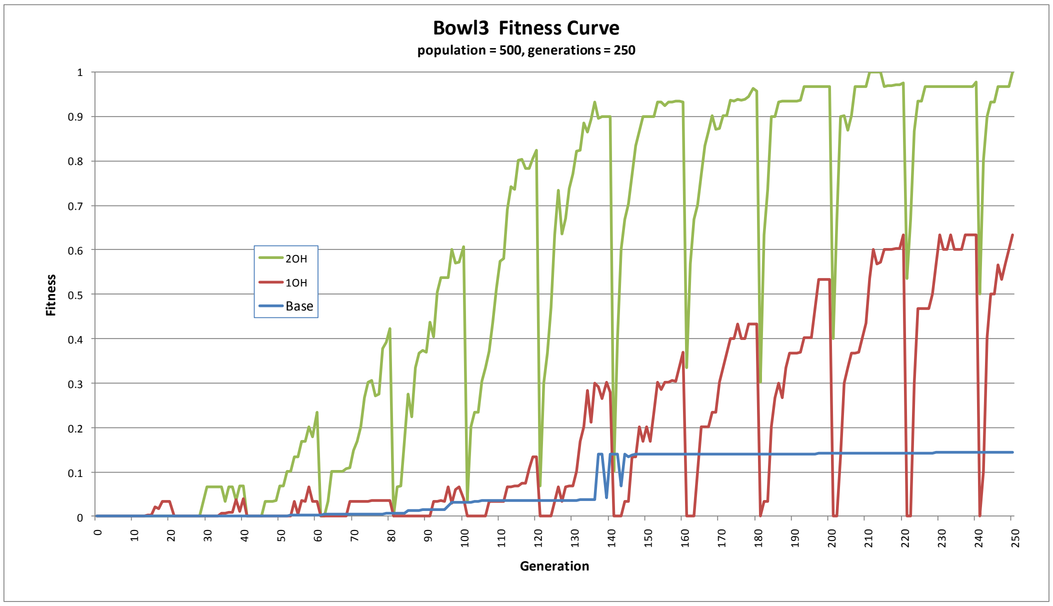


Figure 11 Fitness Curve for the Bowl3 Equation for Base GP, 1st and 2nd Order ACGP

Figure 11 shows the fitness curves for a population of 500 solutions for the original standard GP from Figure 1, ACGP using only 1st order heuristics and ACGP using only 2nd order heuristics. ACGP has performance and solution quality advantage over the standard GP even with this small population. It finds better solutions more quickly. The spikes in the learning curves for ACGP represent the Regrow operator's refreshing of the population after each analysis and adjustment interval.

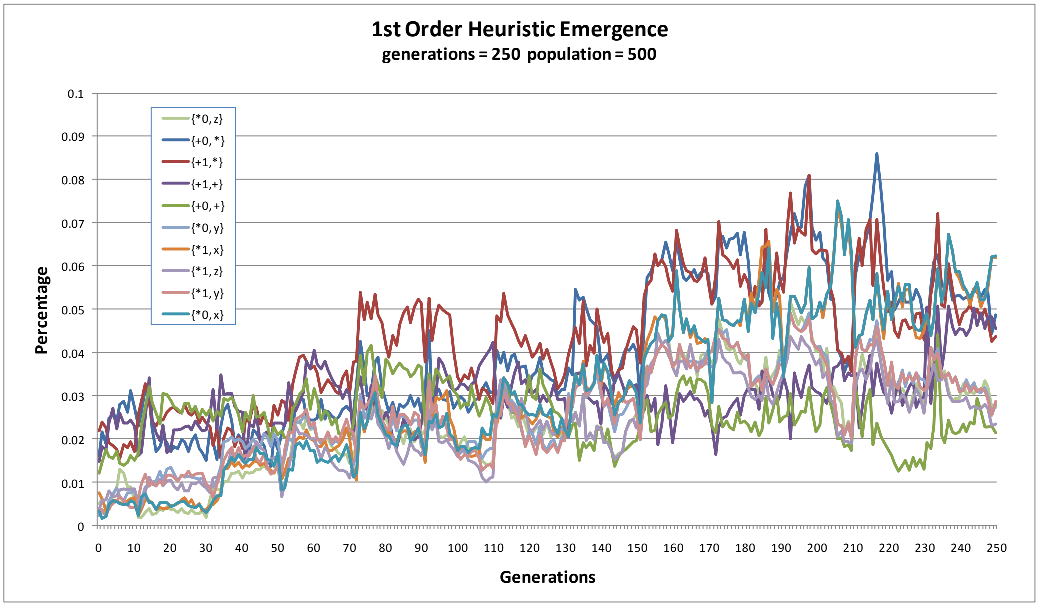


Figure 12 1st Order Building Blocks for Bowl3 (ACGP)

Figure 12 illustrates that ACGP increases the frequency of the desirable 1st order building blocks over a span of generations. The building block frequency percentage curves are nosier than those shown earlier.

Besides the building block frequency variations caused by the standard GP operations, the ACGP Regrow operator induces large frequency shifts by its reinitialization of the population.

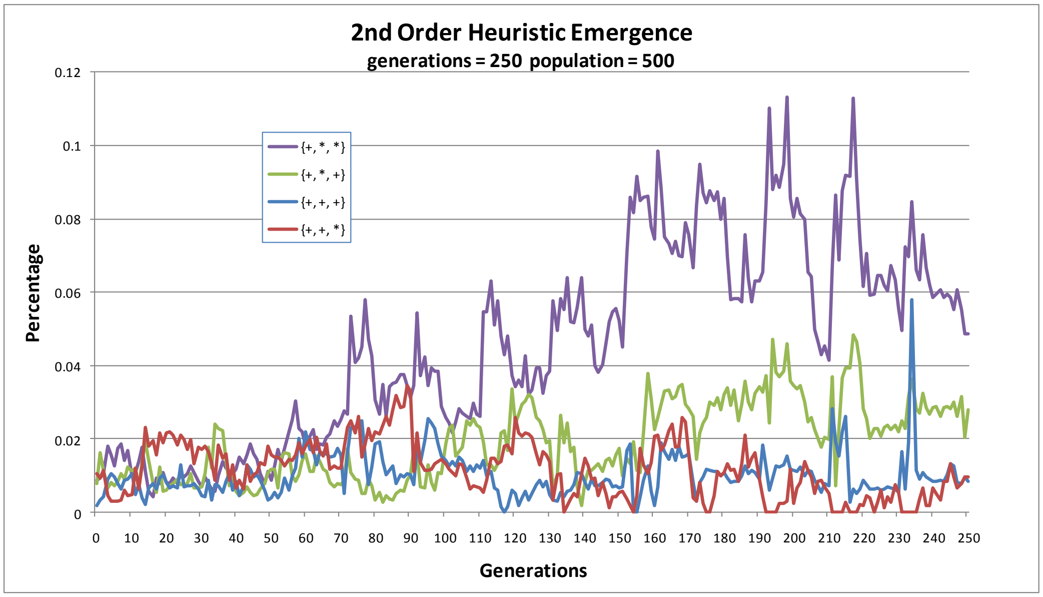


Figure 13 2nd Order building Blocks for Bowl3 (ACGP) (addition root)

Figure 13 shows the ACGP 2nd order building block frequency percentages for subtrees with addition in their root nodes. The desirable building block {+,∗,∗} has a very strong presence in the population while {+,∗, +} is encouraged and its reflection {+, +,∗} is discouraged. The larger frequency turbulence induced by the Regrow operator can easily be seen in this chart. This chart more closely resembles the building block frequency charts of populations larger than the 500 shown here.

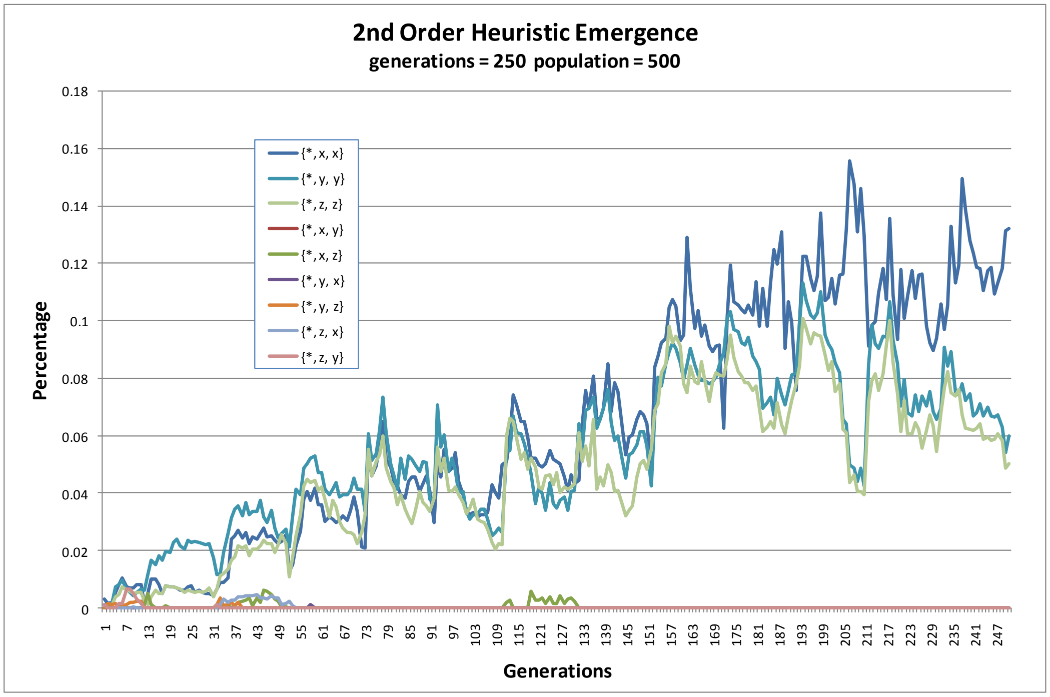
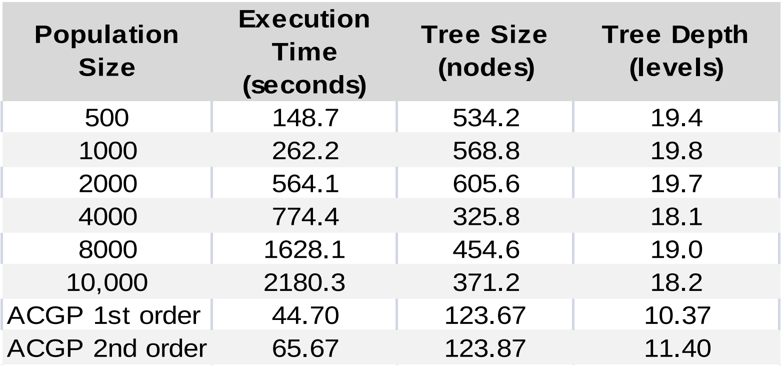


Figure 14 2nd Order building Blocks for Bowl3 (ACGP) (multiplication root)

In the previous 2nd order building block frequency charts for subtrees with multiplication as their root node the desirable building blocks were found but only became dominant in larger populations. Figure 14 shows the ACGP 2nd order building block frequency percentages for subtrees with multiplication in their root nodes. These desirable building blocks are discovered very early and their frequency growth in the population is strong. This demonstrates how well ACGP can discover desirable building blocks and then encourages their propagation by adjusting the probabilities of choosing them.

Table 1 compared the average execution time, tree size and tree depth for various size populations. Other than the execution time – which grew significantly with the increase in population size – all other metrics were not strongly affected by population size. A possible explanation for this behavior can be tree bloat.

Table 2 GP Performance by Population Size



Population Size	Execution Time (seconds)	Tree Size (nodes)	Tree Depth (levels)
500	148.7	534.2	19.4
1000	262.2	568.8	19.8
2000	564.1	605.6	19.7
4000	774.4	325.8	18.1
8000	1628.1	454.6	19.0
10,000	2180.3	371.2	18.2
ACGP 1st order	44.70	123.67	10.37
ACGP 2nd order	65.67	123.87	11.40

Table 2 extends the data from Table 1 by including the same averaged metrics for 30 independent runs of ACGP using 1st order heuristics and ACGP using 2nd order heuristics. In the two ACGP cases there are very strong results that imply that ACGP develops more compact trees. This should not be surprising since ACGP regrows the population after each analysis interval using increasingly weighted building block probabilities. Therefore, in later generations, it will build more compact, higher quality solutions based on the desired building blocks that are discovered in the earlier generations.

These results and other research with ACGP [**11**] hint at the promise of this methodology if extended to higher order building block structures. Unfortunately, this promise may be limited by physical reality. Research focusing on the computational issues associated with the significantly increased combinatorial building block expansion [**10**] has demonstrated that without some additional simplifying manipulations, the analytical building block data structures become large enough that they offset and eventually negate any building block structure information.

Conclusions

Schema theorems for genetic programming offer insight into why GP solve programming problems well. They identify the mechanism which makes this possible – the discovery, propagation and combination of fit building blocks into fit high quality solutions.

These theorems are not without their limitations. They often simplify GP operation to a subset of possible operators to simplify the probabilistic mathematics. They only infer schema growth from one generation to the next and do not extend over the span of several generations, which closer matches the time scale of GP operations.

Nevertheless, some conclusions drawn from GP schema theorems are worth noting. A GP will need a rich initial pool of building blocks for a successful search. Larger populations of solutions provide richer pools of varied building blocks. These larger populations incur a cost of slower execution based on longer time to evaluate the fitness of the population. If a GP does not find desirable building blocks quickly tree bloat works against continued solutions convergence.

The building block frequency research in this paper confirms these conclusions. It shows the advantages of a larger population when using a standard GP. The dramatic shift in building block composition caused by the late emergence of desirable building blocks was an interesting observation from a GP using a small population and reinforces the assertion that it is important to have desirable building blocks emerge early in the GP evolution process.

This paper discussed the results using one methodology that attempts to resolve the two key GP issues: the early discovery of desirable building blocks and control of tree bloat as the population converges to a solution.

Works Cited

1 Altenberg, Lee. The Evolution of Evolvability in Genetic Programming. In Jr., Kenneth E. Kinnear, ed., *Advances in Genetic Programming*. MIT Press, Cambridge, 1993.

2 Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. *Genetic Programming - An Introduction*. Morgan Kaufmann, San Francisco, 1998.

3 De Jong, Kenneth A., Spears, William M., Gordon,Diana F. Using Markov Chains to Analyze GAFOs. In *Proceedings of the Third Workshop on Foundations of Genetic Algorithms* (San Francisco 1995), Morgan Kaufmann, 115 - 138.

4 Goldberg, David E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, 1989.

5 Goldberg, David E., Deb, Kalyanmoy, and Clark, James H. *Genetic Algorithms, Noise, and the Sizing of Populations*. University of Illinois as Urbana-Champaign, Urbana, IL, 1991.

6 Harik, George, Cantu-Paz, Erick, Goldberg, David E., and Miller, Brad L. The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations. In *Evolutionary Computation*. MIT Press, Cambridge, 1997.

7 Holland, John H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

8 Janikow, Cezary Z. *ACGP/CGP lil-gp 1.2;1.02 A User's Manual*. University of Missouri - Saint Louis, Saint Louis, 2008.

9 Janikow, Cezary Z. ACGP: Adaptable Constrained Genetic Programming. In O'Reilly, Una-May, Yu, Tina, and Riolo, Rick L., ed., *Genetic Programming Theory and Practice (II)*. Springer, New York, 2005.

10 Janikow, Cezary Z., Aleshunas, John. Cost-benefit Analysis of Using Heuristics in ACGP. In *2011 IEEE Congress on Evolutionary Computation* (New Orleans, LA 2011), IEEE.

11 Janikow, Cezary Z., Aleshunas, John. Second-Order Heuristics in ACGP. In *ACM Genetic and Evolutionary Computation Conference (GECCO)* (Dublin, Ireland 2011), ACM.

12 Koza, John R. *Genetic Programming*. MIT Press, Cambridge, 1992.

13 Langdon, William B. and Poli, Riccardo. *Foundations of Genetic Programming*. Springer-Verlag, Heidelberg, 2002.

14 Looks, Moshe. *Competent Program Evolution*. Washington University, Saint Louis, 2006.

15 McKay, Robert I., Hoai, Nguyen Xuan, Whigham, Peter Alexander, Shan, Yin, O'Neill, Michael. Grammar-based Genetic Programming: a survey. In *Genetic Programming and Evolvable Machines*. Springer, Berlin, 2010.

16 McPhee, Nicholas Freitag, Ohs, Brian, Hutchison, Tyler. *Semantic building blocks in genetic programming*. University of Minnesota, Morris, Morris, MN, 2007.

17 O'Reilly, Una-May and Oppacher, Franz. The Troubling Aspects of a Building Block Hypothesis for Genetic programming. In Whitley, L. Darrell and Vose, Michael D., eds., *Foundations of Genetic Algorithms 3*. Morgan Kaufmann, Estes Park, 1995.

18 Pelikan, Martin, Goldberg, David E., and Lobo, Fernando. *A Survey of Optimization by Building and Using Probablistic Models*. University of Illinois as Urbana-Champaign, Urbana, IL, 2000.

19 Sastry, Kumara and Goldberg, David E. *Probablistic Model Building adn Competent Genetic Programming*. University of Illinois as Urbana-Champaign, Urbana, IL, 2003.

20 Sastry, Kumara, O'Reilly, Una-May, Goldberg, David E., and Hill, David. *Building-Block Supply in Genetic Programming*. University of Illinois at Urbana-Champaign, Urbana, IL, 2003.

21 Shan, Yin, McKay, Robert, Essam, Daryl, Abbass, Hussein. *A Survey of Probabilistic Model Building Genetic Programming*. University of New South Wales, Canberra, NSW, Australia, 2006.

22 Stephens, Chris and Waelbroeck, Henri. Schemata Evolution and Building Blocks. In *Evolutionary Computation*. 1999.

23 Whigham, P. A. Grammatically-Based Genetic Programming. In *Proceedings of the Workshop on*

Genetic Programming: From Theory to Real-World Applications (Tahoe City, CA 1995).

24 Zongker, Douglas, Punch, Bill, Rand, Bill. *Lil-gp 1.01 Users Manual*. Michigan State University, Ann Arbor, 1996.