5-17-2013

# GP Representation Space Reduction Using a Tiered Search Scheme

John Joseph Aleshunas
*University of Missouri-St. Louis*, jalesh@webster.edu

# GP Representation Space Reduction Using a Tiered Search Scheme

John J. Aleshunas
M.S. Computer Science, Missouri University of Science and Technology, 1994
B.S. Mathematics, Carnegie-Mellon University, 1975

A Thesis Submitted to the Graduate School
at the University of Missouri – St. Louis

In partial fulfillment of the requirements for the degree
Doctor of Philosophy in Applied Mathematics (Computer Science Option)
May 2013

## Abstract

The size and complexity of a GP representation space is defined by the set of functions and terminals used, the arity of those functions, and the maximal depth of candidate solution trees in the space. Practice has shown that some means to reduce the size or bias the search must be provided. Adaptable Constrained Genetic Programming (ACGP) can discover beneficial substructures and probabilistically bias the search to promote the use of these substructures. ACGP has two operating modes: a more efficient low granularity mode ($1^{st}$ order heuristics) and a less efficient higher granularity mode ($2^{nd}$ order heuristics). Both of these operating modes produce probabilistic models, or heuristics, that bias the search for the solution to the problem at hand. The higher granularity mode should produce better models and thus improve GP performance, but in reality it does not always happen.

This research analyzes the two modes, identifies problems and circumstances where the higher granularity search should be advantageous but is not, and then proposes a new methodology that divides the ACGP search into two-tiers. The first tier search exploits the computational efficiency of $1^{st}$ order ACGP and builds a low granularity probabilistic model. This initial model is then used to condition the higher granularity search. The combined search scheme results in better solution fitness scores and lower computational time compared to a standard GP application or either mode of ACGP alone.

## Acknowledgements

There are so many people I am indebted to for helping me complete this thesis.

I would specifically like to thank my advisor, Cezary Z. Janikow, whose knowledge, insights, and wisdom have been invaluable at various times during the past four years. As a strong empiricist he forced me to be clear and rigorous in my work. He tolerated my theoretical focus and kept pulling me back to the application reality. This taught me discipline and focus.

I want to thank my dissertation committee members, Sanjiv Bhatia, Uday Chakraborty, and Wenjie He. Because of their diverse research interests, each has contributed important breath to my education. I value the perspective this has given me.

I owe a large intellectual debt to Daniel St. Clair, my Master's advisor and mentor. Dr. St. Clair showed me the wonders of machine learning and its application to real-world problems. He started me on this path but reached the end of his life before I began this stage. I miss his saying, "If we knew what we were doing, we wouldn't be able to call it research."

Finally, and most importantly, I want to thank my wife, Pamela Aleshunas. She encouraged me, she supported me, and, when I had doubts, she pushed me to believe in myself. I want to thank her so very much for her continued unwavering support in all that I do.

## Table of Contents

## List of Figures

## List of Tables

# Dissertation

## 1 Background

Problem solving is a central theme in mathematics and computer science. Individuals devote considerable effort to the development of algorithms for automated problem solving. Deterministic algorithms work well when the problem space is tractable or the search space lends itself to a deterministic guided search. Once the search space gets large, deterministic methods experience time and processing costs that overwhelm them [16]. Heuristic search techniques are potentially more efficient and productive in these situations [4], [16]. Heuristic search of very large and complex problem domains using concepts from Darwin's theory of evolution offers an interesting option. Many evolution-based search methodologies are inspired by the selection behavior and survival outcomes in nature. As early as 1948, Turing proposed "genetical or evolutionary search" [4]. Today there are multiple nature-inspired types of evolutionary computation (EC): evolutionary programming, genetic algorithms (GA), evolutionary strategies, evolutionary computing, and genetic programming (GP). All of these techniques are based on the Darwinian principles and evolve a set of candidate solutions guided by a fitness evaluation function.

The GP methodology differs from the other evolutionary computation schemes. Where most of the EC methods are typically applied to optimization problems, GP applications are more similar to machine learning [4]. The goal of a GP application is to evolve a set of computer programs and automatically improve them based on their fitness to accomplish a given task [3]. In other words, a GP application is an evolutionary search method that is adept at solving optimal instruction set problems [12]. A GP implementation strives to improve a set of computer programs using feedback from experiences with problem domain data.

Genetic programming often represents its population of candidate solutions as variable-sized trees. This choice of representation is mainly due to the early GP research work of John Koza [12] and this discussion will assume this solution model. The solution trees in GP applications are composed of elements from a predetermined set of functions and terminals. The function set consists of the operators, functions, and statements of the GP problem domain [12]. The members of the function set have some number of arguments. Members of the set of functions and terminals can be assigned to these argument locations. The terminal set consists of the constants supplied to the GP application and the variables representing data inputs [12]. These components are called terminals because they have no arguments, as functions do, and only appear in the leaf nodes of the population trees [3].

Several principles guide the choice of functions and terminals for a particular GP implementation. It is important that the set of functions and terminals be complete enough to represent a solution to the given problem [12]. This is referred to as the *sufficiency property*. If a necessary function or terminal is missing, the GP implementation may have difficulties finding a solution or only be able to find suboptimal solutions. On the other hand, if the set of functions and terminals are too large it will generate a large and complex search space thereby impeding the probability of finding a viable solution. Finally, it is assumed that the members of the functions set accept any member of the function or terminal sets as valid arguments to avoid problems with invalid labelings [12]. This is called the *closure property* [3]. While this property is preferred, it cannot always be accommodated. Closure is not a problem when the members of the set of functions and the output values of the functions in the function set are all the same data type. If different data types are used in a GP application then constraints or other rules are necessary to support the closure property [12].

Solutions are evolved from existing population members using genetic operators such as *crossover*, *mutation*, and *reproduction*. *Crossover* generates new solutions by exchanging genetic material, or subtrees, between two parent solutions. This process can be both constructive and destructive [12], [13]. When the operator combines simpler elements into a more successful complex structure, it is constructive. Crossover can also tear apart successful solutions. This latter behavior is destructive and can inhibit the search process. *Mutation* randomly modifies a sub-tree of a solution to create a new candidate solution. Like crossover, mutation can be both constructive and destructive [12], [13]. Normally, crossover and mutation are controlled using uniform probabilities and therefore they comprise an unbiased genetic search. Some methods that depart from this model result in interesting GP behavior. *Reproduction* is the simplest GP operator. It copies a *selected individual* into the new population unchanged [3], [12].

*Selection* is the principal driver in most GP implementations [3], [12]. This mechanism chooses fit population members for participation in the generation of the next population of solutions and induces a bias toward fitter solutions. Selection works best when the fitness function provides graded and continuous evaluation of how well the GP search is progressing [3]. *Fitness proportional selection* exerts a strong fitness bias to the search process. Other selection schemes, such as *tournament selection*, moderate this bias, often in order to mitigate problems associated with premature convergence on a suboptimal solution [3]. The choice of selection method as a parameter can be used to improve the quality of the search and the fitness results of a conventional GP implementation for a given problem.

The search quality and fitness results of a conventional GP application can vary with different choices for a number of parameters, not just selection. Often these parameter choices differ from problem to problem [12]. Some typical parameters whose adjustment can potentially impact GP search are: population size, selection method, operator probabilities, and initial and maximal tree size [3]. A larger population of candidate solutions increases the diversity of genetic material available at the cost of computing resources. Population diversity should help breed better candidate solutions sooner. Selection methods, such as tournament selection, that are not as greedy as fitness proportional selection prevent premature convergence and maintain diversity in the population of candidate solutions [3], [4]. Some problems cannot be described by smaller tree structures. Adjusting the minimum tree size for the initial population reduces the number of potential unfit or invalid solutions and potentially increases the number of viable solutions in the population [3]. Normally, fitness evaluation is the most computationally intensive operation in a GP application. Often fitness samples are designated as a percent of the total population size. This value can be adjusted when large populations are used in a GP implementation to prevent oversampling. This adjustment can improve fitness convergence in large populations [3]. While proper set of parameters can improve GP search for specific problems, this benefit often imposes offsetting computational or storage costs and does not directly address the underlying issue – the size and complexity of the representation search space.

The next chapter will identify a foundational issue that impacts GP performance – size and complexity of the representation space, along with a survey of the several proposed methods to deal with those issues by using or building some problem models. Chapter 3 will identify another method that this research attempts to improve, along with identifying the cases when it fails to deliver. Then, Chapter 4 proposes a new strategy to improve that method by using a two-tiered approach. The next chapter details the new strategy and illustrates its effectiveness with a number of experiments. Finally, Chapter 6 presents the conclusions of the research presented here along with a discussion of topics where future research work is required.

## 2 Characterization of the GP Representation Space

One critical issue with GP application 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 representation space becomes so large that the search of this large space impacts the performance of the GP application [3]. Methods that can help discover the best subset of functions and terminals thereby reducing this representation space should improve the performance and solution quality of a GP implementation.

My previous work [10] quantified the combinatorial growth of the representation space in terms of the function elements, the terminal elements, the arity of the functions, and the maximum number of levels permitted in population individuals.

$$M(level) = \sum_{i}^{F_i} (M(level - 1) + |T|)^{arity_{F_i}} \quad where\ M(0) = |F| \qquad (1)$$

Equation (1) recursively calculates the size of the total set of trees from one level of nodes up to a maximum size identified by the variable *level.* This space contains every combination of the functions *F,* the terminals *T*, and the arity of each function $arity_{Fi}$ for every tree of depth 1 up to the stated maximum depth. The size for a set of zero-size trees (only root nodes) is simply the magnitude of the terminal set and these are normally not viable solutions for any non-trivial problem. Experimentation with this equation quickly demonstrates that the population for this representation space will grow exponentially for a given set of functions and terminals as the maximum level of trees is increased. As an example of the scope of this problem, Table 1 lists the population growth for a GP component set using four binary functions and 14 terminals.

**Table 1 - GP Application Representation Space Growth**

| Level | Population Size |
|---|---|
| 1 | $1.29 \times 10^{3}$ |
| 5 | $8.07 \times 10^{58}$ |
| 10 | $4.96 \times 10^{1903}$ |
| 15 | $8.49 \times 10^{60936}$ |
| 20 | $2.46 \times 10^{1950000}$ |

It is easily seen that the extremely large search spaces defined by large multi-level trees normally used in GP applications complicate our solution search. Genetic algorithms do not display as high a level of complexity as GP applications because they have a simpler alphabet of components (normally only two characters), position specific semantics, and they often use fixed-sized population members.

Table 2 compares the growth of the representation space for a GP application using four binary functions and 14 terminals versus another GP implementation using five binary functions and 14 terminals. The inclusion of only one more binary function has an impact on the representation space growth which increases as the maximal tree level is increased. This table implies that if the set of functions and terminals is reduced to a minimal set the resulting representation space will be smaller than that generated by any super set of functions and terminals.

**Table 2 - GP Application Representation Space Growth (4 versus 5 binary functions)**

| Level | 4 Functions | 5 Functions |
|---|---|---|
| 1 | $1.29 \times 10^{3}$ | $1.81 \times 10^{3}$ |
| 5 | $8.07 \times 10^{58}$ | $4.38 \times 10^{61}$ |
| 10 | $4.96 \times 10^{1903}$ | $1.61 \times 10^{2026}$ |
| 15 | $8.49 \times 10^{60936}$ | $2.13 \times 10^{64860}$ |
| 20 | $2.46 \times 10^{1950000}$ | $1.41 \times 10^{2075553}$ |

Table 3 compares the representation space size of a GP implementation using four binary functions and 14 terminals versus another GP implementation using three binary functions plus one tertiary function and 14 terminals. The substitution of the tertiary function for one of the binary functions has a highly significant impact on the size of the resulting representation space. This table implies that if the set of functions and terminals is reduced to a minimal set using functions with fewer arguments rather than more arguments, the resulting representation space will be smaller than the space generated by any set of functions that include higher arity functions. If the results of a higher arity function can be generated by combinations of lower arity functions it is computationally simpler to not use the higher arity function. Clearly, any reduction in the elements of the function set or terminal set can help reduce the representation search space.

**Table 3 - GP Application Representation Space Growth (4 binary functions versus 3 binary + 1 tertiary functions)**

| Level | 4 Binary Functions | 3 Binary + 1 Tertiary Functions |
|---|---|---|
| 1 | $1.29 \times 10^{3}$ | $6.80 \times 10^{3}$ |
| 5 | $8.07 \times 10^{58}$ | $3.40 \times 10^{310}$ |
| 10 | $4.96 \times 10^{1903}$ | $1.33 \times 10^{75459}$ |
| 15 | $8.49 \times 10^{60936}$ | $1.49 \times 10^{18336567}$ |

The discussion above about the GP application representation search space ignores some specific attributes of the search space components that can help reduce the complexity and thereby improve the GP search. One technique that can help constrain the size of the search space is restricting the solution generation process so that it can only generate semantically valid individuals for the given problem. One such validation technique utilizes the argument type requirements for the member functions to constrain which functions or terminals are valid as child nodes. Some functions require specific types of functions or terminals in particular argument locations. An example of such a function is the logical IF function. Normally IF is defined as a function with three arguments: the first argument requires a Boolean function or terminal, the other two arguments can normally be any function or terminal. Restricting which functions or terminals are selected for the first argument to the IF function can guarantee that only valid solutions are generated by a GP application. There may be a need to impose additional constraints on the operation of a GP application, this notion of enforcing particular GP implementation structure constraints is the concept behind Strongly Typed GP and Grammar-Based GP.

## 2.1 Strongly Typed GP

Strongly Typed GP (STGP) strives to satisfy the GP closure requirement by constraining the selection of functions and terminals in a GP solution based on their data type for particular argument locations [20]. In STGP, every terminal is assigned a type and every function has types assigned to each of its arguments and its return value. These types then help constrain the random selection of functions and terminals so that only valid solutions are generated. These type constrains govern the initial generation of population members, locations and subtrees involved in crossover, and subtree generation for mutation.

The STGP methodology is effective in improving the GP search because computational resources are not wasted on evaluating invalid solutions. Another positive aspect of these structural constraints is that they reduce the effective search space. The search space is reduced to only the valid solutions that can be constructed from the set of functions and terminals. For specific sets of functions and terminals, this space reduction can be significant. Consider the impact of STGP constraints on the search space size computation in Equation (1). The equation assumes that every function and terminal component is valid at any location in an individual tree. The STGP rules constrain which function and terminal components are valid for a given parent node function and the particular parent node function argument these components describe. These rules constrain both the initial trees and how operators generate new trees so that only valid trees are generated. Many of the individual trees that an unconstrained Equation (1) would count are not included in the search space of a STGP. The argument type rules are normally set prior to the start of the GP search for many STGP implementations [20]. This requirement is more of a reality of this technique rather than a limitation.

## 2.2 Grammar Based GP

Similar to STGP, Grammar-based GP implementations were developed to constrain the structure of generated solutions so that only valid solutions are generated and evaluated [15], [20], [24]. The most common grammar formalism is context-free grammars (CFG) [23]. Grammars are a natural way to express solution generation constraints. A grammar consists of a set of rewrite or production rules that govern the combinations of function and terminals. Here is an example of a grammar that enforces the structure for a particular GP problem:

$$tree ::= E \times sin(E \times t)$$

$$E ::= var \mid (E\ op\ E)$$

$$op ::= + \mid - \mid \times \mid \div$$

$$var ::= x \mid y \mid z$$

These grammar production rules ensure that the initial population consists of only valid individuals. Additionally, these rules guide the process of the GP operators of crossover and mutation. The extra cost of processing these rules is offset by the elimination of invalid solutions in the grammar-based GP search space and the resulting search space can be significantly smaller than the space generated by random permutations of the selected function and terminal sets for a given GP problem. This smaller search space consisting of only valid structures helps improve the search process of a grammar-based GP.

When the constraints of a grammar are considered in the context of Equation (1), the reduction of the representation space complexity is clear. The equation assumes that every function and terminal component is valid at any location in an individual tree. The grammar rules dictate which function and terminal components are valid for a given parent node function and the particular parent node function argument these components describe. Many of the individual trees that an unconstrained Equation (1) would count are not included in the search space of a CFG-GP.

Originally, CFG-GP required that the grammar rule sequence be specified prior to execution of the GP search [23]. This restriction can impose a potential negative limitation on the GP search, and possibly ignore an optimal solution, when more than one set of the grammar rules is valid or when the grammar is unknown. Stochastic context-free grammar (SCFG) is an approach that attempts to avoid this restriction [23]. In a SCFG the production rules are stipulated in advance in a specific production sequence, similar to CFG. The difference in SCFG is that each rule is assigned a probability. Normally these probabilities start as a uniform distribution and the rules are applied in the original sequence to produce individual solutions for a population. As future GP implementation generations are evaluated, the production rule probabilities are adjusted. These probability adjustments bias the sequence of production rules toward a sequence that produces fitter individuals and modify the sequence of the production rules. SCFG offer adaptability in the probabilities of the set of production rules but inferring an SCFG is a difficult problem [23]. Most current SCFG methods are based on a greedy search scheme.

## 2.3 EDA-GP

Estimation of distribution algorithm GP (EDA-GP) is another approach to resolving the complexity of the GP representation search space [23]. EDA-GP models the search space as a probability distribution and is based on EDA which were developed to solve genetic algorithm problems. Probabilistic incremental program evolution (PIPE) is an example of an EDA-GP technique [17], [19], [20], [21]. PIPE replaces the GP population with a hierarchy of probability tables organized into a tree structure (see Figure 1). Each table represents the probability that a particular function or terminal will be chosen for that particular location in a newly generated individual solution.

PIPE simplifies the representation space search problem by not directly searching the representation space [17]. It generates a sample of search space using the probability model, updates the probability distribution based on the fittest solutions in this sample, and repeats this process until a stopping criterion is met. In each generation, individual solutions are bred using the probability tables at each node. A function or terminal primitive is chosen for a particular location based on that node's probability table. Once a sample population is generated, the fitness of these new programs is computed. Finally, the probability table hierarchy is updated using the sample population fitness. The goal of this process is to make the future generation of highly fit solutions more likely. In PIPE each node is treated as an independent random variable. PIPE attempts to learn the probability of particular functions at particular locations in the tree and therefore will ignore other potentially successful solutions as the options for particular nodes emerge [17].

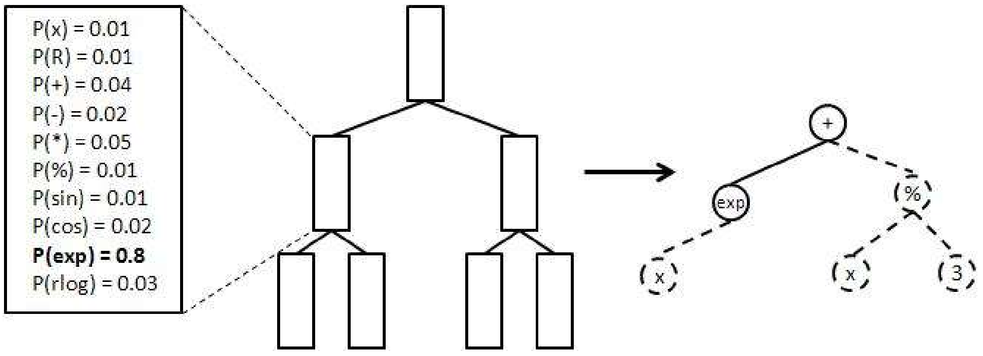

**Figure 1 - Probabilistic incremental program evolution (adopted from [23])**

While PIPE avoids a direct struggle with the GP representation space complexity, there are several nuances of this methodology that should be noted. The probability table hierarchy tree must be sized large enough to capture the depth of a viable solution tree. Next, each node's probability table must include a probability value for every function of terminal in the set of primitives for the GP problem. Finally, the branches from each node to its child nodes are dominated by the arity of the function with the highest arity [17]. All of these three structural requirements must be stipulated at the start of the GP search and together they impose a different form of complexity in the operation of PIPE.

One advantage of the PIPE methodology is that each node probability table is localized within the hierarchal tree of probability tables. This means that each individual function or terminal can have a different probability depending on its depth or location in the tree. One outcome of this behavior is that functions are more probable higher in the tree and terminals are more probable as the tree's depth increases [20].

The search space defined by Equation (1) assumes that every function and terminal component is valid at any location in an individual tree and has a non-zero probability of being selected. PIPE (and most EDA-GP) alters the selection probability of the function and terminal components for each node in an individual tree but normally does not restrict a component from being selected by reducing this probability to zero. This behavior means that while some individuals may have a higher probability of being generated, any individual in the entire search space can still be generated. The complexity of the representation search space is probabilistically constrained, similar to SCFG, rather than being physically constrained as in STGP and CFG-GP.

The individualized nature of the selection process at each node is also a limitation of PIPE. Since the functions and terminals are selected for each node independent of any of the other nodes, PIPE cannot capture any dependencies among the elements in the function and terminal set. Another limitation of this methodology stems from the requirement that the probability table hierarchy tree must be sized large enough to capture the depth of a viable solution tree. This requirement imposes a physical memory cost to store the probability table hierarchy tree [20] and the cost quickly increases as the maximal tree size and function arity increases.

## 2.4 Modular Models

Highly complex objects in biological or engineering domains often use hierarchical, modular structures to mitigate their complexity. This concept is easily adapted to GP solution structures and has been a topic of early research in genetic programming.

The building block hypothesis [3] asserts that highly fit building blocks combine to form highly fit individuals in the GP population from one generation to another. These highly fit components are threatened by the destructive effects of the crossover operator. Early work by Koza [12] proposed the modularization of program code or a subtree into an automatically defined function (ADF). With ADF, a GP implementation can evolve potentially reusable components that are available for use in GP population individuals. Recombination operators are constrained within ADF so that an ADF subtree is swapped only with another individual's ADF subtree [20].

ADF have shown success in several problem domains and they provide a performance advantage when the introduction of functions will sufficiently reduce the length of solutions in a population. A limitation of ADF is the requirement to define the architecture of an individual program prior to executing a GP implementation using ADF. The ADF architecture must stipulate the number of function-defining branches in the whole program and the number of arguments for each function-defining branch. This requirement increases the set of initial parameters for the GP application and the complexity of the process.

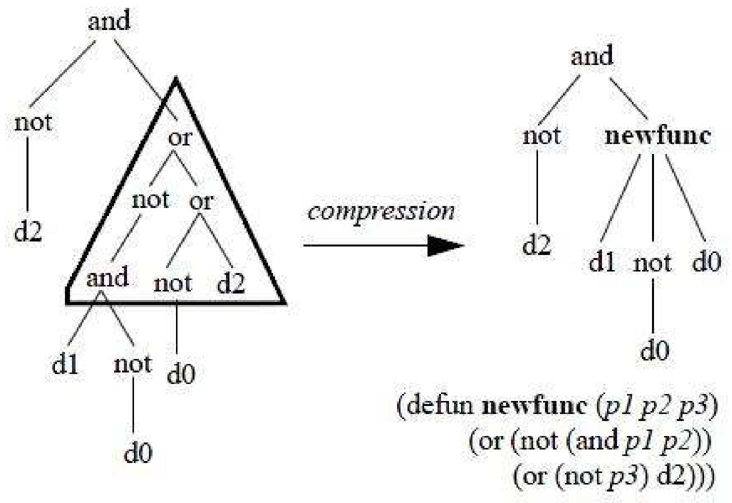

**Figure 2 - Modular Model GP (adopted from [2])**

This concept of modular models was extended by Angeline and Pollack [2]. In their approach, each module is stored in a library of modules and referenced in the individuals of the GP population. Modules that provide a fitness advantage will be referenced more frequently as the number of fit individual solutions increases. Modules that are no longer referenced probably contribute little to the fitness of individuals and can be culled from the library. This technique can help constrain the physical storage needed for the population of solutions but adds a third category of node components – a set of function modules. Figure 2 shows an example where a subtree is converted to a reusable module.

If these additional function modules are considered in Equation (1) along with the set of functions and terminals, the complexity of the representation space can be described by Equation (2).

$$M(level) = \sum_{i}^{F_i} (M(level-1) + |T| + |Modules|)^{arity_{F_i}} \quad where\ M(0) = |F| \qquad (2)$$

This equation can be simplified. Some modules will not have any arguments and terminate a tree branch similar to the formal terminals. These modules can be combined with the terminals to form an augmented terminal set $T'$. Other modules have arguments and can be combined with the original functions to form an augmented function set $F'$. These augmented sets produce this reformulation of Equation (1).

$$M(level) = \sum_{i}^{F'_i} (M(level-1) + |T'|)^{arity_{F'_i}} \quad where\ M(0) = |F'| \qquad (3)$$

While Equation (3) resembles Equation (1), its results are in fact different because the augmented set of functions and terminals reduce the number of levels necessary in candidate solution trees. Overall, modular models have several advantages. They can discover building blocks that contribute to the fitness of individual trees and encapsulate them in the module library for reuse. They also protect those building blocks as modules from the destructive effects of crossover and mutation. These advantages are offset by the augmented set of functions and terminals described by Equation (3) and the additional work of searching for modules.

## 2.5 Meta-optimizing Semantic Evolutionary Search

Another probabilistic approach was proposed by Looks [14]. The *meta-optimizing semantic evolutionary search* (MOSES), while not a GP methodology, is an evolutionary process that attempts to find optimal programs by optimizing the representation space of a given problem. The operation of MOSES is based on the premise that if an optimal representation space is defined for a given programming problem, a solution should be easily found. MOSES attempts to refine the representation space parameters for a given problem using the Hierarchical Bayesian Optimization Algorithm (hBOA) [18]. Some examples of these representation space parameters that MOSES tries to optimize are: the set of functions and terminals, numerical constants that modify the behavior of a function, weighted combination of terminals rather than use of a single terminal, or relative weighting of the arguments of a function.

The operation of MOSES is a repeating two stage process shown in Figure 3. Starting with an initial representation space parameterization, MOSES generates the initial population of programs. It then chooses a set of highly fit programs as a sample for representation analysis. The representation analysis step uses the competent optimization algorithm hBOA to refine representation space parameters. These new parameters are used to generate new candidate programs that are added to the population; possibly replacing less fit programs. These stages of sampling, representation refinement, and population renewal are repeated until an optimal program is generated.

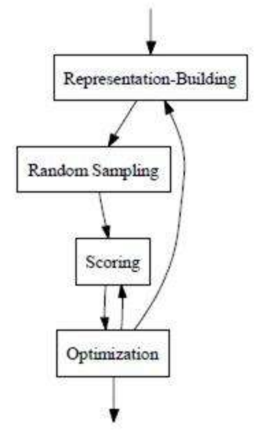

**Figure 3 - Operational Overview of MOSES (adopted from [14])**

While not a typical genetic programming methodology, MOSES is relevant to this discussion because of its process of representation space refinement. Its operation is designed around the concept that optimization of the representation space will eliminate the generation of non-productive individuals and thereby improve the ease of finding an optimal solution. By optimizing the representation space, MOSES reduces the size of the searched regions of the representation space. This search space reduction improves the methodology's search capabilities and the quality of the search results.

Each of the GP variants discussed above utilizes constraint techniques to either constrain or condition the GP representation space and thereby improve learning. Strongly typed GP and context-free grammar GP use explicit constraints that are specified prior to beginning the GP search process. Stochastic context-free grammar GP and estimation of distribution GP develop probabilistic constraints/models as the GP search progresses. Modular models attempt to discover highly fit components and foster their reuse as learning proceeds. Meta-optimizing semantic evolutionary search attempts to optimize the parameters that define the representation space and thus improve the GP search of that space. The next chapter will explore and detail operations of another similar methodology, Adaptable Constrained Genetic Programming, which can use explicit constraints and also build a probabilistic model. This methodology will be analyzed and then extended here.

# 3 Constrained Genetic Programming and Adaptable Constrained Genetic Programming

Constrained Genetic Programming (CGP) [5] is a method that attempts to constrain the components and combinations of functions and terminals used in a particular GP application. CGP was originally developed to ensure that only valid solutions were generated in a GP search by designating which functions and terminals were valid for any given function argument position. The permissible, and/or prohibited, arguments are fed to CGP at start up and applied unchanged throughout an execution. These constraints can be either strong (absolute) constraints or weak (probabilistic) constraints. Strong CGP constraints stipulate which functions or terminals can be used (or not used) in a particular argument location. Weak CGP constraints assign probabilistic weights for each function and terminal in particular argument locations. Later empirical work [9] demonstrated that when proper CGP constraints are used, the GP application conducts a more efficient search and solves the problem at hand faster. The limiting factor of this method was that any reduction in the set of functions and terminals presented to start a CGP run required a trial and error discovery process as shown by Janikow and Mann [9]. CGP was then extended into Adaptable Constrained Genetic Programming (ACGP) [7], [8] so that the process of discovering which functions and terminals and which combinations of functions and terminals were needed to better solve a particular GP problem could be automatically discovered during an ACGP run.

## 3.1 The Operational Aspects of ACGP

This description of the operation of ACGP is more detailed than the discussion of the previous methods. The search methodology described in this dissertation is an improvement of ACGP therefore an understanding of ACGP's operation is necessary to appreciate of the proposed methodology.

Initially, ACGP tracks and adjusts the probability that a particular function or terminal component will be used in a given function argument location. This is done separately for the root location, and for all other locations independent of position. These heuristics capturing the probability of function and argument pairs are tracked in ACGP frequency tables. These frequencies are then used to update the actual probabilities.

The basis for the methodology that tracks and adjusts these heuristics is grounded in the GP Building Block Hypothesis [13]. The GP Building Block Hypothesis asserts that GP operators induce fit low-order building blocks to combine and form higher-order building blocks, eventually converging to optimum or near-optimum solutions. While this is an appealing concept, it is criticized from the basis that the fitness of individual building blocks cannot be normally assessed outside of the context of a whole solution [13]. This behavior is caused by the inability to decompose most problems into subcomponents whose fitness can be directly measured. ACGP uses a methodology that infers the fitness of individual building blocks without actually computing that fitness [6]. During the heuristic assessment processing, ACGP tabulates the frequency of each building block in the fittest population individuals. It uses an assumption that those building blocks that are highly frequent in fit solutions contribute to the fitness of those solutions. The tabulated frequency counts are then used to adjust the heuristic weights that govern the selection that a particular building block will be used – the frequencies can be used as new probabilities, but in practice the frequencies are slowly combined with previous weights to modify the probabilities.

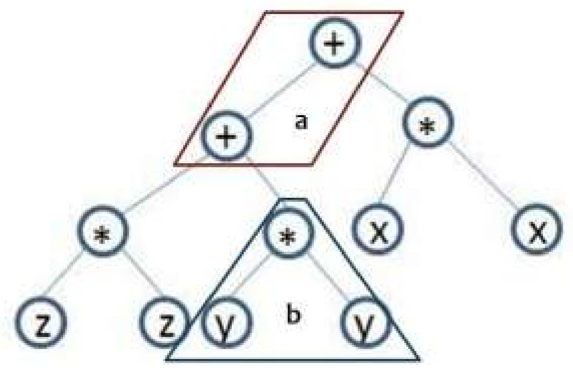

**Figure 4 - Examples of 1st and 2nd Order Building Blocks**

Those heuristics that appear frequently in highly fit population members are reinforced and all other heuristics are suppressed. As these probabilities are adjusted, the search space is probabilistically constrained and thus the search is also constrained. ACGP has a set of operating parameters that control the rate of adjustment. As mentioned, adjustments can be applied at a constant rate or using a linearly increasing rate. Initially ACGP worked with 1st order heuristics: the probability of a function or terminal will be chosen for one of a function's arguments (block (a) in Figure 4) [7]. 1st order ACGP discovered better information of the beneficial structures in the representation space versus a standard GP while also being computationally efficient [8], [9]. This concept was later extended to work with 2nd order heuristics: the probability a combination of functions or terminals will be chosen to be the set of arguments of a parent function (block (b) in Figure 4) [11]. 2nd order ACGP increases the granularity of beneficial structure information but this improvement comes at the cost of additional computation and storage [10], [11]. Further extension of this method to 3rd order or greater heuristics is plagued by the combinatorial issues discussed in [10] and are therefore computationally infeasible or at least ineffective.

Once the heuristic weights are adjusted, ACGP can generate an entirely new population with a *regrow* operator using these improved heuristics. This new regrow operator, introduced in ACGP [6], enhances the problem space search by restarting the population with an improved set of solutions. An additional benefit of the regrow is the suppression or minimization of introns and solution bloat [11]. Each new population is regrown using the starting tree size parameters for the specific GP problem implementation. Each heuristic assessment cycle continues to refine and improve the heuristic weights. These improving weights are used for, and bias the operation of, crossover, mutation, as well as regrow thereby making ACGP behavior in line with SCFG and different than a normal GP application where crossover and mutation are uniformly random rather than biased.

A simple designed regression example will help illustrate the advantages of ACGP. The Bowl3 equation (Equation (4)) is intentionally designed to demonstrate ACGP search versus a standard GP. Additionally, its building block design highlights the informational advantage 2nd order ACGP has over 1st order ACGP. Figure 5 shows how well ACGP improves over a base GP implementation using either 1st order heuristics or 2nd order heuristics. This example problem shows the fitness results averaged for 30 independent runs solving a regression problem for the equation:

$$Bowl3 = (x * x) + (y * y) + (z * z) \qquad (4)$$

The training length for all three techniques in these experiments spans 500 generations. The downward spikes visible in the 1st order and 2nd order ACGP curves are caused by the regrowing of the population following the recurring heuristic assessment process at stipulated iterations (every 20 generations, in this example). The regrown populations exhibit better and better initial fitness following the regrow operation because each successive cycle uses a better set of heuristic selection weights.

The informational learning advantage that ACGP has over a standard GP is clear. As 1st order ACGP discovers better heuristics, the heuristic information helps bias crossover, mutation, and regrow to use better building block combinations and therefore build better candidate solutions. The larger heuristics in 2nd order ACGP increase its information regarding better building block combinations and productive search regions in the GP representation space. This additional information helps improve 2nd order ACGP search and thereby its candidate solutions as seen in Figure 5.

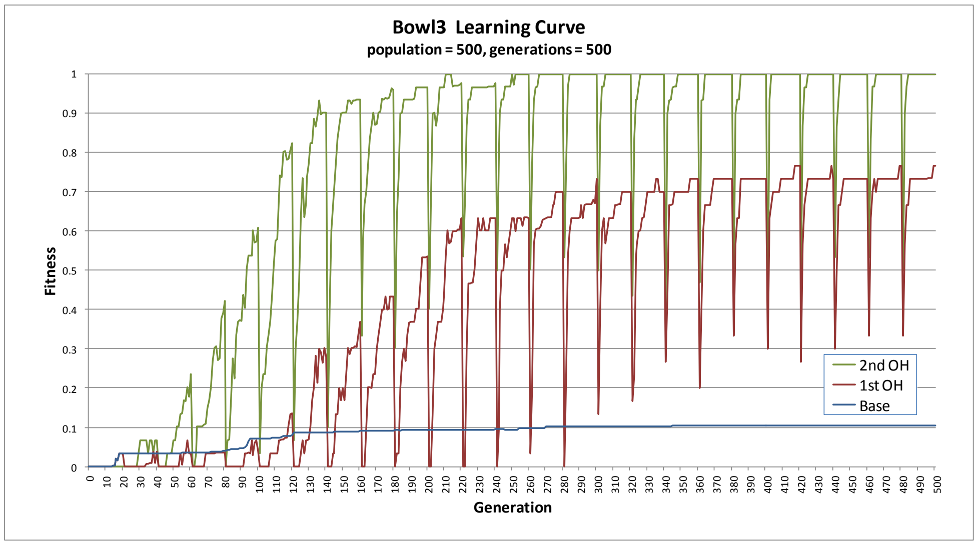

**Figure 5 - ACGP versus Base GP Learning Curve for Equation (4) using a Population of 500 (from [11])**

All of the GP searches in the experiments shown in Figure 5 above used the following operational parameters:

**Table 4 - GP Operational Parameters**

| | |
|---|---|
| Function set | $\{+, -, *, /\}$ (protected divide) |
| Terminal set | {x, y, z, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5} |
| Population size | 500 |
| Generations | 500 |
| Operators | crossover 85%, mutation 10%, reproduction 5%, regrow 100% at each iteration |
| Iteration length | 20 generations |
| Selection method | Tournament, using a tournament size of 7 |
| Number of independent runs | 30 |
| Fitness | sum of square errors on 100 random data points in the range -10 to 10 adjusted to a range of 0 to 1 with 1 being best (When tracing fitness, the best solution at each generation from the 30 independent runs was averaged) |
| ACGP heuristic adjustments | Made using a linearly increasing rate based on the analysis of fittest 10% of the population. The rate is such that after x% of generations, the observed frequencies replace x% of the previous probabilities. This is referred to as a sloped training. |

All of the experiments in this document use the same parameters as listed in Table 4. Any variations will be identified in the discussion of the particular experiment.

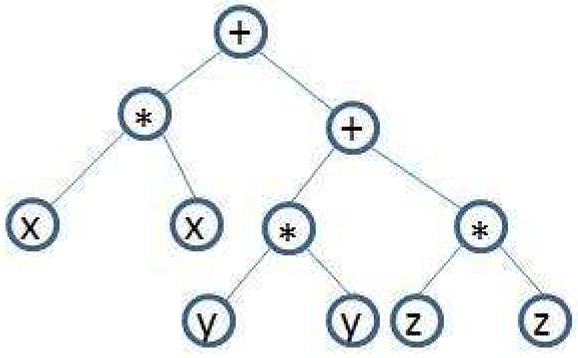

**Figure 6 – Example Solution Trees for Equation (4) (*Bowl3* Equation)**

Figure 6 depicts an example of a minimal solution trees for this problem. Since the solution equation uses only variable-structure functions (‘+’ and ’*’), multiple tree structures yield equivalent fitness evaluations.

Equation (4) (*Bowl3*) was specifically designed for this experiment because it is a good exemplar of the benefits of ACGP search - it possesses a particular building block structure in its target equation. This structure emphasizes the advantage of 2nd order heuristics over 1st order heuristics. The simplicity of this regression example makes it easy to demonstrate the search advantages incorporated in ACGP. The target equation is composed of ten desirable 1st order building blocks:

$$\begin{gathered}\{+_1 +\}, \{+_2 +\}, \{+_1 *\}, \{+_2 *\}, \\ \{*_1\, x\}, \{*_1\, y\}, \{*_1\, z\}, \{*_2\, x\}, \{*_2\, y\}, \{*_2\, z\}\end{gathered} \tag{5}$$

This notation shows each 1st order building block enclosed by braces. The parent function is on the left. The subscript of the parent function designates the argument location. The child element at the given argument location is designated on the right. For example, $\{+_1 *\}$ means + function having its 1st (left) argument labeled with the ‘*’ function. The 1st order building blocks in Equation (5) *implicitly* describe thirteen 2nd order building blocks:

$$\begin{gathered}\{+ + +\}, \{+ + *\}, \{+ * +\}, \{+ * *\}, \\ \{x * x\}, \{x * y\}, \{x * z\}, \\ \{y * x\}, \{y * y\}, \{y * z\}, \\ \{z * x\}, \{z * y\}, \{z * z\}\end{gathered} \tag{6}$$

This notation shows each 2nd order building block in infix form enclosed by braces. The parent function is in the middle of the set of elements. It is flanked on either side by its children. However, not all of these implicit 2nd order building blocks are desirable for our regression solution to Equation (4). Only six of these implied building blocks are necessary to form fit solutions (using the minimum solution trees in Figure 6):

$$\begin{gathered}\{+ + *\}, \{+ * +\}, \{+ * *\}, \\ \{x * x\}, \{y * y\}, \{z * z\}\end{gathered} \tag{7}$$

ACGP running only in 1st order mode processes 2nd order heuristics implicitly by putting together its 1st order heuristics, but the actually desired 2nd order heuristics can be different yet impossible to generate in the 1st order mode. This *differential* between the 2nd order building blocks implied by the 1st order building blocks and the explicit desirable 2nd order building blocks needed for fit solutions is the informational advantage 2nd order ACGP has over 1st order ACGP. Additionally, this differential is also reflected in both the 1st order and the 2nd order ACGP heuristic probabilities.

As an example, consider the heuristics for function ‘*’ and argument *x*. There are three possible building blocks for the first argument of ‘*’ and three building blocks for the second argument of ‘*’ (using the image in Figure 6) but only one of each has *x*. Assuming perfect heuristic probabilities, the two 1st order building blocks ($\{*_1\, x\}$ and $\{*_2\, x\}$) will each have an ideal heuristic probability of 0.333 in 1st order ACGP. When running in 1st order ACGP, the implied 2nd order building block $\{x * x\}$ , produced by putting together the above two 1st order heuristics, will have a combined implicit probability of 0.111. When the explicit 2nd order building blocks are considered directly in 2nd order ACGP, $\{x * x\}$ appears once out of three explicit '*' building blocks; therefore the explicit $\{x * x\}$ 2nd order heuristic will actually have an ideal heuristic probability of 0.333 in 2nd order ACGP search. This heuristic probability differential (implied with 0.111 versus explicit with 0.333) emphasizes the informational advantage of 2nd order ACGP over 1st order ACGP. This differential is quite large and it shows when both modes are compared in Figure 5.

This is an interesting observation that can be verified by experimenting with another regression problem that does not exhibit any differential. Consider the equation:

$$Bowl3full = (x * x) + (x * y) + (x * z) + (y * x) + (y * y) + (y * z) + (z * x) + (z * y) + (z * z) \quad (8)$$

Equation (8) is an interesting counter example for this discussion. This equation is constructed such that it is composed of the same ten desirable 1st order building blocks that define Equation (4) (shown in Equation (5)). These 1st order building blocks combine to imply the same thirteen 2nd order building blocks as Equation (4) (shown in Equation (6)). The difference between Equation (8) and Equation (4) is that all of these implied 2nd order building blocks are actually desirable for our regression solution to Equation (8) and there is no information differential between the 2nd order building blocks implied by the desired 1st order building blocks and the actual explicit 2nd order building blocks in Equation (8).

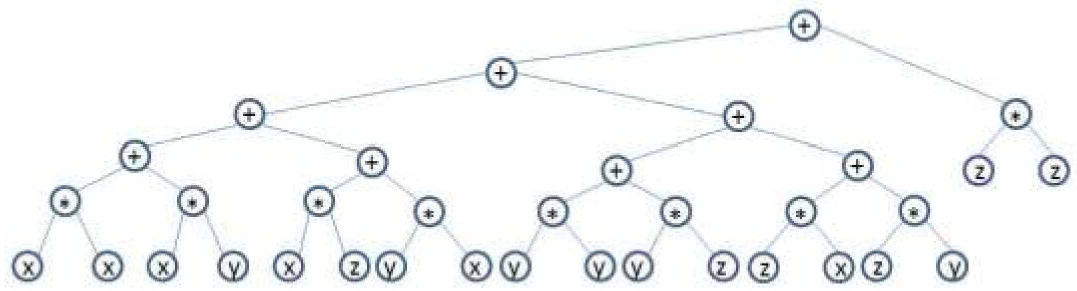

**Figure 7 – An Example Solution Tree for Equation (8) (*Bowl3full* Equation)**

Figure 7 depicts an optimal solution tree for the *Bowl3full* problem (Equation (8)). Since the solution equation uses only variable-structure functions ('$+$' and '$*$'), multiple tree structures yield equivalent fitness evaluations. The desired building blocks should be easily discovered and, because of the variable-structure functions in the optimal solution, they can appear freely in the candidate solutions with equivalent fitness scores. All of the GP searches in the experiments shown in Figure 8 used the same operational parameters identified in Table 4.

Consider the same heuristics for function '*'. Using the example solution in Figure 7, there are three 1st order building blocks for the first argument of '*' ($\{*_1\ x\}, \{*_1\ y\}, and\ \{*_1\ z\}$) and three 1st order building blocks for the second argument of '*' ($\{*_2\ x\}, \{*_2\ y\}, and\ \{*_2\ z\}$). Assuming perfect heuristic probabilities, the two 1st order heuristics for the variable $x$ ($\{*_1\ x\}$ and $\{*_2\ x\}$) will each have and ideal heuristic probability of 0.333 in 1st order ACGP. Combining these two 1st order building blocks to form the implied 2nd order building block $\{x * x\}$ produces again a combined heuristic probability of 0.111. When the explicit 2nd order building blocks are considered directly in 2nd order ACGP, $\{x * x\}$ appears once out of nine explicit '*' building blocks, therefore the explicit $\{x * x\}$ 2nd order heuristic will actually have an ideal heuristic probability of 0.111 in 2nd order ACGP search. This lack of heuristic probability differential (implied with 0.111 versus explicit with 0.111) shows zero informational advantage of 2nd order ACGP over 1st order ACGP. This lack of an information differential between the implicit and explicit 2nd order building blocks would predict that 2nd order ACGP should have no advantage over 1st order ACGP.

Figure 8 shows the fitness learning curve for Equation (8) averaged over 30 independent runs. The ACGP methodology retains its advantage over a basic GP implementation because it discovers and adjusts the selection probabilities of the desirable 1st and 2nd order building block. Compared to Equation (4) and Figure 5 these 1st and 2nd order ACGP results show two important differences:

- As speculated above, due to lack of differential, there is no difference between the two modes in solving the problem (other than initially)
- ACGP does not do as well with Equation (8) compared to its results with Equation (4)

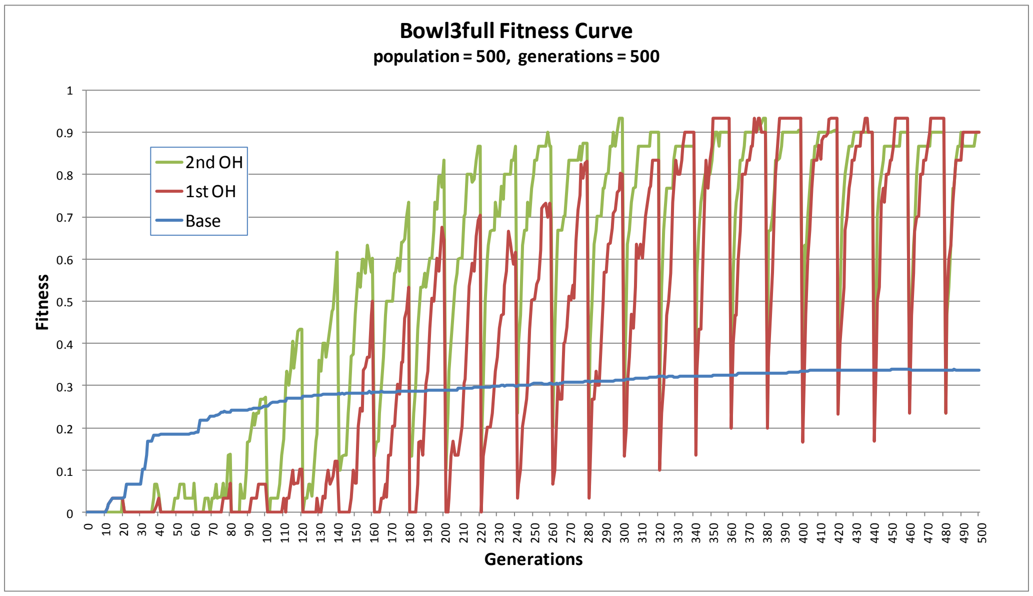

**Figure 8 - ACGP versus Base GP Learning Curve for Equation (8) using a Population of 500**

While successful in its current configuration, ACGP, like other GP methods, can have difficulties finding a solution to problems with complex structure. Assuming some differential does exist, the 2nd order mode should perform better than the 1st order mode, but they both may suffer due to specific problem characteristics – these building blocks may now be harder to find.

## 3.2 The Difficulties of ACGP Search

Most functions used in GP applications can be organized into two general classes of function types: variable-structure functions and strict-structure functions. Variable-structure functions are functions where the order of the arguments does not change the result of the function's evaluation. Examples of variable-structure functions are: arithmetic *addition*, *multiplication* and logical *AND* and *OR*. A GP solution composed of variable-structure functions can evaluate to the same fitness value using many different structure permutations. Strict-structure functions, in contrast, evaluate to different values if the order of the arguments is changed. ACGP's heuristic adjustment mechanism can both exploit and be deceived by the strict tree structure dictated by strict-structure functions. The efficacy of the heuristic adjustment process is dependent on the density of quality candidate solutions in the sample used to analyze the building blocks. The goal of the ACGP heuristic adjustment mechanism is to increase the selection probability of fit building blocks and suppress the probability that less fit building blocks are selected. Unfortunately, as stated in the previous chapter, the fitness of individual building blocks cannot be determined separately from the fitness of the individual population member. The ACGP heuristic adjustment mechanism is based on an assumption that if a building block occurs frequently in fit population members then it possibly contributes to the high fitness of those individuals and therefore it is assumed to be a fit building block [6], [7], [11]. This assumption driving the ACGP heuristic adjustment mechanism works well when relatively fit solutions are sampled for heuristic adjustment. The analysis and adjustment process is more successful when the target solution consists of primarily variable-structure functions which therefore have a higher probability of sampling highly fit solutions. Alternatively, when the target solution consists of more strict-structure functions, the structure of a highly fit candidate solution becomes more rigid and there is a lower probability of ACGP sampling a fit solution for heuristic analysis. When ACGP samples solutions with poor fitness it continues to bias the heuristic weights based on the discovered building block frequencies. Regrettably, these highly frequent building blocks are not the elements of fit solutions and therefore their heuristics will induce potentially random solutions and not fit solutions.

As an example, consider the regression problem for the following equation:

$$ComplexEq = (x * x * x) + (5 * x * y) - (3 * y * z * z) - (y * y * y) \qquad (9)$$

While this problem is more complex than Equation (4), it also differs structurally by the inclusion of the '–' function as well as variable-structure functions ('$+$' and '$*$'), and scalar constants. This equation is designed to demonstrate a specific evolutionary search induced by complex types of problems. This function induces a rigid structure on part of the solution tree. This rigid tree structure requires the positioning of large subtrees in specific locations of a successful candidate solution (Figure 9) while ACGP does not deal with specific locality of its heuristics.

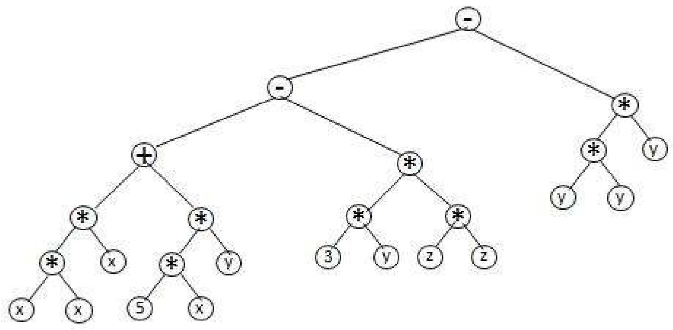

**Figure 9 – An Example Solution Tree for Equation (9) (*ComplexEq*)**

The structural complexity of Equation (9) is also reflected in both the 1st order and 2nd order building blocks that create a viable candidate solution. Equation (9) is composed of seventeen desirable 1st order building blocks:

$$\begin{gathered}\{*_1 *\},\{*_2 *\},\{+_1 *\},\{+_2 *\},\{-_1 +\},\{-_1 -\},\{-_2 *\},\\ \{*_1 x\},\{*_2 x\},\{*_1 y\},\{*_2 y\},\{*_1 z\},\\ \{*_2 z\},\{*_1 3\},\{*_2 3\},\{*_1 5\},\{*_2 5\}\end{gathered} \tag{10}$$

These 1st order building blocks combine to describe thirty-nine implicit 2nd order building blocks:

$$\begin{gathered}\{* * *\},\{* * x\},\{* * y\},\{* * z\},\{* * 3\},\{* * 5\},\\ \{x * *\},\{y * *\},\{z * *\},\{3 * *\},\{5 * *\},\\ \{x * x\},\{x * y\},\{x * z\},\{x * 3\},\{x * 5\},\\ \{y * x\},\{y * y\},\{y * z\},\{y * 3\},\{y * 5\},\\ \{z * x\},\{z * y\},\{z * z\},\{z * 3\},\{z * 5\},\\ \{3 * x\},\{3 * y\},\{3 * z\},\{3 * 3\},\{3 * 5\},\\ \{5 * x\},\{5 * y\},\{5 * z\},\{5 * 3\},\{5 * 5\},\\ \{* + *\},\{+ - *\},\{- - *\}\end{gathered} \tag{11}$$

Not all of these building blocks are desirable for a GP regression solution to Equation (9). Only twenty-one 2nd order building blocks are necessary to form fit solutions:

$$\begin{gathered}\{* * *\},\{* * x\},\{* * y\},\\ \{x * *\},\{y * *\},\\ \{x * x\},\{x * 5\},\{y * y\},\{y * z\},\{y * 3\},\{y * 5\},\\ \{z * y\},\{z * z\},\{z * 3\},\\ \{3 * y\},\{3 * z\},\{5 * x\},\{5 * y\},\\ \{* + *\},\{+ - *\},\{- - *\}\end{gathered} \tag{12}$$

Consider the '*' heuristics for the variable $x$ in an example solution such as Figure 9. There are six unique building blocks for the first argument of '*' ($\{*_1 *\},\{*_1 x\},\{*_1 y\},\{*_1 z\},\{*_1 3\}, and\ \{*_1 5\}$) out of nine total first argument building blocks. There are four unique building blocks for the second argument of '*' ($\{*_2 *\},\{*_2 x\},\{*_2 y\}, and\ \{*_2 z\}$) out of nine total second argument building blocks.

Assuming perfect heuristic probabilities, the two 1st order building blocks for the variable $x$ ($\{*_1\ x\}\ and\ \{*_2\ x\}$) will each have an ideal heuristic probability of 0.111 and 0.333 in 1st order ACGP. Combining these two 1st order building blocks to form the implied 2nd order building block (processed in 1st order), $\{x * x\}$ produces a combined heuristic probability of 0.037. When the explicit 2nd order building blocks are considered directly (Figure 9), $\{x * x\}$ appears once out of nine explicit '*' building blocks, therefore the explicit $\{x * x\}$ 2nd order heuristic will actually have an ideal heuristic probability of 0.111 in 2nd order ACGP. This small heuristic probability differential (implied with 0.037 versus explicit with 0.111) shows a slight informational advantage of 2nd order ACGP over 1st order ACGP. This small differential between the implicit 2nd order building blocks and the explicit 2nd order building blocks would indicate an informational advantage for ACGP while using 2nd order heuristics in learning Equation (9). However, something else happens in the experiment.

Figure 10 shows the fitness learning curve for Equation (9) using a population of 1000 individuals with averaged fitness scores of 30 independent runs for a base GP, 1st order ACGP, and 2nd order ACGP. While the fitness scores are not impressive (note the modified y-axis scale), the base GP application outperforms both ACGP methods, and due to bad performance we cannot see any potential advantage of 2nd order ACGP.

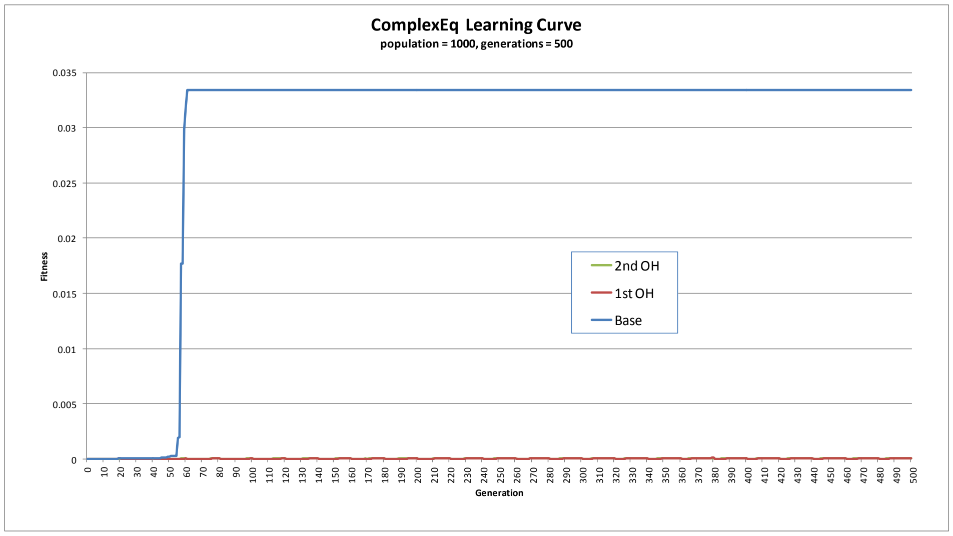

**Figure 10 – ACGP versus Base GP Learning Curve for Equation (9) using a Population of 1000**

The relatively small population of this experiment and the low probability of sampling fit solutions work against ACGP's ability to discover desirable heuristics regardless of mode or differential. Even with a population of weak solutions, ACGP will tabulate the most frequent building blocks and enhance their probability of selection [1]. Unfortunately, without a sample of good solutions, this behavior becomes deceptive. ACGP will continue to tabulate the most frequent building blocks, irrespective of their contribution to solution fitness, and adjust their heuristic probabilities accordingly. These poor heuristics lead the ACGP search toward poor solutions and this behavior explains the ACGP fitness results in Figure 10.

Increasing the population size is a normal parameter modification that can improve the GP search for a problem like this one. As mentioned in Chapter 1, using a larger population is a typical method used to increase the population diversity and potentially produce a sample of viable solutions. Figure 11 shows the learning curve for Equation (9) using a population of 7000 individuals with averaged fitness scores of 30 independent runs. Both modes of ACGP now outperform the base GP application but 2nd order ACGP still does not perform better than 1st order ACGP despite the stated small differential. The increase in the population size helps ACGP sample better solutions. It is not a surprising result that a larger GP population will sample potentially better solutions for heuristic analysis [22]. The fact that the 2nd order ACGP does not outperform 1st order ACGP is an interesting observation – while we know it should. While this larger population provides a good sample of solutions for both ACGP modes, the population increase is not sufficient here to allow 2nd order ACGP to utilize its differential to outperform 1st order ACGP. Further increasing population size will decrease efficiency especially due to slower performance of 2nd order ACGP (further discussed later). What is clearly needed is a method to utilize higher population with the more efficient 1st order ACGP discovering useful coarser granularity heuristics and then using this information to bias the 2nd order ACGP to utilize the differential.

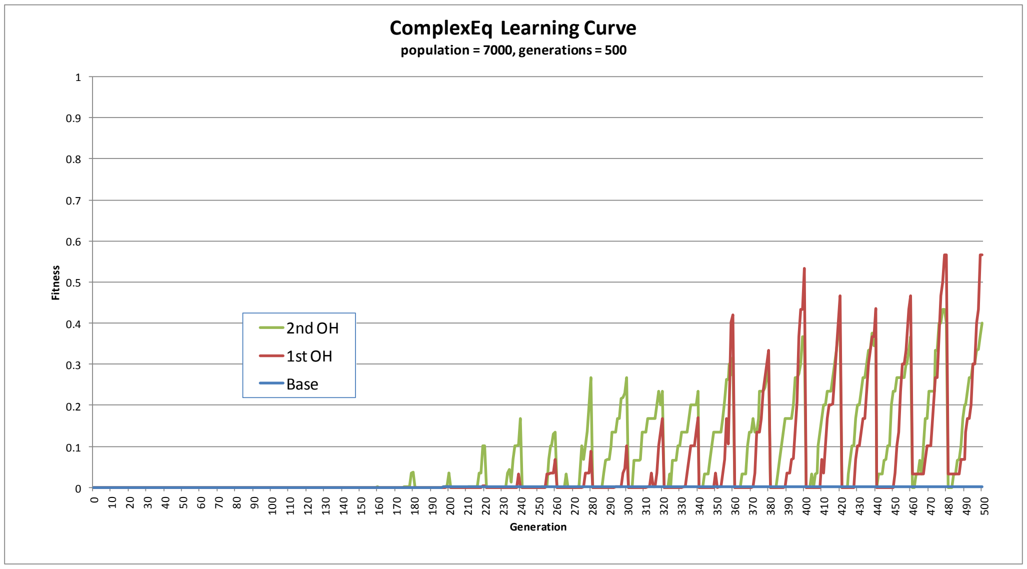

**Figure 11 - ACGP versus Base GP Learning Curve for Equation (9) using a Population of 7000**

# 4 Statement of Research Thesis

Consider a simple thought experiment. Before leaving the house, a man goes searching for his coat. He will assign some search areas a higher search priority than others; based on how likely successful he believes they will be based on, for example, past searches. While two high-priority search areas may be physically distant from each other, he treats them as being adjacent, skipping over the low-probability areas between them. This methodology reduces the complexity of his search space and helps him find his jacket quicker.

This thought experiment, practiced over the years in many Artificial Intelligence approaches, suggests a method to find and promote desirable building blocks in an ACGP search. The quality of a search can be improved if it is constrained by a probabilistic map of the search space that reflects a productive search history, much like the search for a jacket described above. The characteristics of each ACGP mode suggest an operating scheme that can capitalize on the advantages of both modes of ACGP and mitigate their shortcomings.

Normally, the ACGP search starts with uniform probability of each heuristic and is dependent on the quality of the solutions in the sample of individuals under ACGP analysis. 1st order ACGP is less deceived by complex or strict structured problems than 2nd order ACGP because it searches for lower granularity building blocks. This advantage stems from the smaller number of 1st order building blocks versus the larger number of 2nd order building blocks for a given set of functions and terminals. This lower complexity translates into a more computationally efficient search for 1st order ACGP. On the other hand, 2nd order ACGP offers a means of exploiting any granularity differential. A two-tiered search methodology using the two ACGP operating modes can exploit the advantages of each mode and mitigate their disadvantages. The first tier would search the representation space using 1st order ACGP. This search will result in a probabilistic map of the 1st order heuristics. The heuristic map will bias the use of productive building blocks and discourage the use of less productive ones when it is used to bias the 2nd order ACGP search in the second tier of the method. The second tier search should be more efficient and attain better results because it starts in a pre-conditioned state that is biased toward more efficient regions of the representation space.

The goal of this research will be the improvement of ACGP learning using a two-tiered search process. This method will develop a probabilistically constrained model of the representation search space using a less granular 1st order ACGP search in an initial phase. Then it will use that model to discover a highly fit solution using a more granular 2nd order ACGP search on the constrained space. This two-tiered scheme should improve the combination of computational efficiency and fitness learning quality over standard 1st order or 2nd order ACGP alone.

# 5 A Tiered ACGP Search Methodology

The discussion in Chapter 1.1 demonstrated mathematically that the complexity of a GP search space is a function of the set of functions and terminals selected to develop a solution and the reduction of this set will reduce this complexity and improve the search process. While this statement appears to be a simple solution that can reduce the search space complexity, there is a problem in choosing which function or terminal components can be removed or its probability adjusted. Normally, the fitness of individual components cannot be easily evaluated in isolation from complete individual solutions. Without some method of evaluating the contribution of individual components to the overall fitness of candidate solutions, component pruning will be a blind process.

The description of ACGP in Chapter 3 demonstrated that ACGP can discover desirable 1st order and 2nd order building blocks. After conducting normal GP training for a set interval of generations (an iteration), ACGP adjusts the probability of the selection of particular building blocks for use in crossover, mutation and regrow. This method to promote the use of desirable building blocks conversely reduces the probability of using non-desirable building blocks. This building block probability adjustment process develops a probabilistic map of the representation search space that is biased toward some solution structures and against others. ACGP 2nd order heuristics, when successful, provide more granular information about the desirable building block structures for a given problem [11]. This differential advantage over ACGP 1st order heuristics is offset by longer computation time, increased data structure storage and processing requirements [10], and difficulties to properly sample the larger set of heuristics result in a potential to underperform ACGP 1st order heuristics in complex problems as demonstrated in the discussion of Chapter 3.2.

Since ACGP allows the input of a heuristic profile to condition it's starting heuristics prior to initiating the search, a proposed tiered search methodology seems to be a possible option that could improve ACGP search. All of the experiments using this two-tier scheme will use the ACGP parameters described in Table 4 unless noted otherwise. Each baseline set of experiments using the base GP application, 1st order ACGP, and 2nd order ACGP will be run using two different population sizes. One population will match the larger population size of the 1st tier processing. The other population will match the smaller size of the 2nd tier processing. The rationale for these population sizes is explained in Chapter 5.3 following a small proof-of-concept experiment to demonstrate the proposed methodology.

## 5.1 A Simple Experiment – the *Bowl3neg* Problem

While previously shown Equation (9) demonstrates how a strictly structured target solution makes the GP search more difficult, its complexity induces several issues that combine to mask any problems that ACGP may have in a particular search. A simpler problem designed to exhibit strict structural behavior without additional complexity may help illuminate the nuances of ACGP search. Equation (13) (*Bowl3neg*) is a variation of previous Equation (4) designed to specifically demonstrate a strict structure solution isolated from the benefits of ACGP search. Equation (13) differs from Equation (4) by substituting '–' for '+'.

$$Bowl3neg = (x * x) - (y * y) - (z * z) \qquad (13)$$

Figure 12 depicts an example optimal solution tree for this problem. Since this target equation includes '–' as well as a variable structure function ('*'), different tree structures will yield different fitness evaluations. The use of a strict-structure function restricts the structure of viable solution trees and forces the other subtree structures into specific locations in a viable candidate solution.

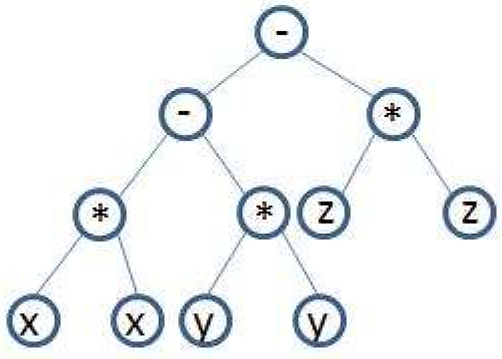

**Figure 12 – An example Solution Tree for Equation (13) (*Bowl3neg*)**

Equation (13), like Equation (4), is designed to be a good exemplar of the benefits of ACGP search because it possesses the same particular building block structure differential that emphasizes the advantage of 2nd order heuristics over 1st order heuristics found in Equation (4). The primary difference between this equation and Equation (4) is the substitution of the '–' function for the '+' function in this equation. This substitution imposes a strict structure that limits the number of possible tree permutations that have a top fitness score. Like Equation (4), this equation is composed of ten desirable 1st order building blocks:

$$\begin{gathered}\{-_1 -\}, \{-_2 -\}, \{-_1 *\}, \{-_2 *\}, \\ \{*_1\, x\}, \{*_1\, y\}, \{*_1\, z\}, \\ \{*_2\, x\}, \{*_2\, y\}, \{*_2\, z\}\end{gathered} \qquad (14)$$

Similar to Equation (4), these 1st order building blocks combine to produce thirteen implicit 2nd order building blocks:

$$\begin{gathered}\{- - -\}, \{- -*\}, \{* - -\}, \{* - *\}, \\ \{x * x\}, \{x * y\}, \{x * z\}, \\ \{y * x\}, \{y * y\}, \{y * z\}, \\ \{z * x\}, \{z * y\}, \{z * z\}\end{gathered} \qquad (15)$$

Not all of these 2nd order building blocks are desirable for our regression solution to Equation (13). Only six of these building blocks are necessary to form a fit solution, which mimics the behavior of Equation (4).

$$\begin{gathered}\{- -*\}, \{* - -\}, \{* - *\}, \\ \{x * x\}, \{y * y\}, \{z * z\}\end{gathered} \qquad (16)$$

The differential between the 2nd order building blocks implied by the 1st order building blocks and the explicit desirable 2nd order building blocks is the informational advantage 2nd order ACGP has over 1st order ACGP.

The 1st and 2nd ACGP heuristic probability analysis for this problem is similar to the analysis of the *Bowl3* problem (Equation (4)). Consider the '*' heuristics for the variable $x$. There are three building blocks for the first argument of '*' and three building blocks for the second argument of '*' in the optimal solution tree in Figure 12. Assuming perfect heuristic probabilities, the two 1st order building blocks for the variable $x$ ($\{*_1\ x\}\ and\ \{*_2\ x\}$) will each have and ideal heuristic probability of 0.333 in 1st order ACGP search. Combining these two 1st order building blocks to form the implied 2nd order building block $\{x * x\}$ produces a combined heuristic probability of 0.111. When the explicit 2nd order building blocks are considered directly '*' with $x$ as both of its arguments appears once out of three explicit '*' building blocks, therefore the explicit $\{x * x\}$ 2nd order building block will actually have an ideal heuristic probability of 0.333 in 2nd order ACGP search. This heuristic probability differential (implied with 0.111 versus explicit with 0.333) emphasizes the informational advantage of 2nd order ACGP over 1st order ACGP. The primary difference with this regression problem compared to the *Bowl3* problem (Figure 6) is that the building blocks containing the terminal elements for this problem must appear in a solution tree in specific locations relative to each other for them to contribute to the solution fitness while ACGP does not process any location information Most alternative building block positions will evaluate to lower fitness scores. This characteristic reduces the frequency of viable solutions in the overall representation space and therefore reduces the probability of finding viable solutions in a given population sample making the search more complex for both ACGP operating modes.

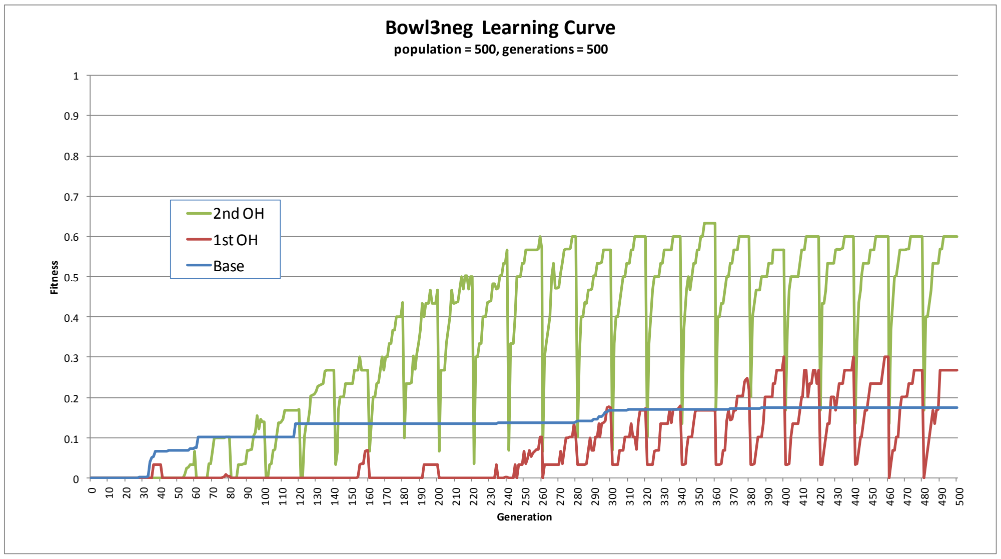

**Figure 13 - Bowl3neg (Equation (13)) using a Population of 500**

Figure 13 shows the fitness learning curves for the Base GP, ACGP 1st Order, and ACGP 2nd Order for the *Bowl3neg* regression problem using the same GP parameters (Table 4) as the experiments in Figure 5. The advantage that the ACGP 2nd order heuristics have over the 1st order heuristics is clearly visible in this chart. This benefit is produced by the differential between the desired 2nd order building blocks and the full set of 2nd order building blocks implied by the desired 1st order building blocks. This differential is similar in magnitude to the one in the *Bowl3* 2nd order building blocks and therefore it implies a similar advantage. What are noteworthy in this regression problem are the lower fitness learning curves for both 1st order and 2nd order ACGP heuristics. These average fitness scores demonstrate the impact of the strict structure solutions in the search space for *Bowl3neg* versus the variable structure of viable solutions in the search space for *Bowl3* has on the ACGP fitness learning for these problems.

As discussed earlier, use of a larger population is a parameter that can be adjusted to improve the GP search. It can increase the diversity in the GP population at the cost of greater fitness computation time [3], [12]. Figure 14 shows the learning curve for a set of experiments using a population of 1000 to find a solution for Equation (13). These experiments exhibit an increased average fitness for all three methods compared to Figure 13.

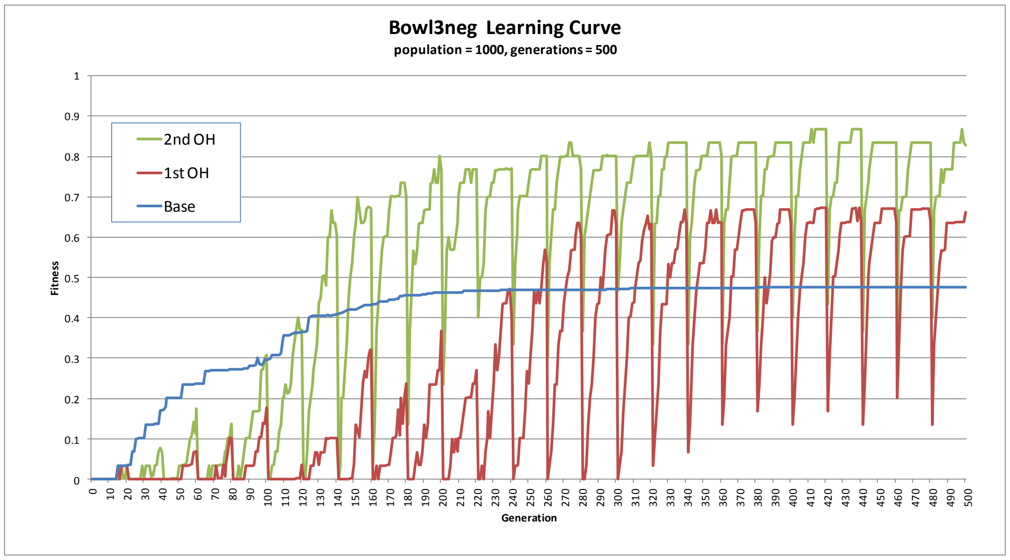

**Figure 14 - *Bowl3neg* Learning Curve using a Population of 1000**

Table 5 demonstrates that an increase of the population size will also increase the execution time. This table also shows that 1st order ACGP is faster than 2nd order ACGP using the same set of environmental training parameters (population size, maximum generations, training parameters, etc.). This is an advantage for 1st order ACGP that may be a benefit in a modified approach.

These results suggest a technique that may mitigate the overall cost of training with a larger population and improve the final results. ACGP 1st order could search the problem space using a large population for a lower number of generations. The results of that search would then be used to condition an ACGP 2nd order search using a smaller population. This scheme would exploit the generality of the low granularity search capability of 1st order ACGP. Moreover, the 1st order ACGP preconditioning could improve the more granular 2nd order ACGP search. The combined efficiencies of this tiered approach might yield productive results.

**Table 5 - Comparison of Average Execution Times (*Bowl3neg*)**

| Execution Times | Population Size | |
|---|---|---|
| | 500 | 1000 |
| Base GP | 272.83 | 411.77 |
| ACGP 1st OH | 46.77 | 88.28 |
| ACGP 2nd OH | 68.90 | 113.83 |

The following experiments use the GP and ACGP parameters shown in Table 4 with the following exception:

> The two-tiered method uses a population of 1000 individuals for the first stage and 500 individuals for the second stage. The transition between the two stages is at 250 generations with both tiers adding up to 500 generations.

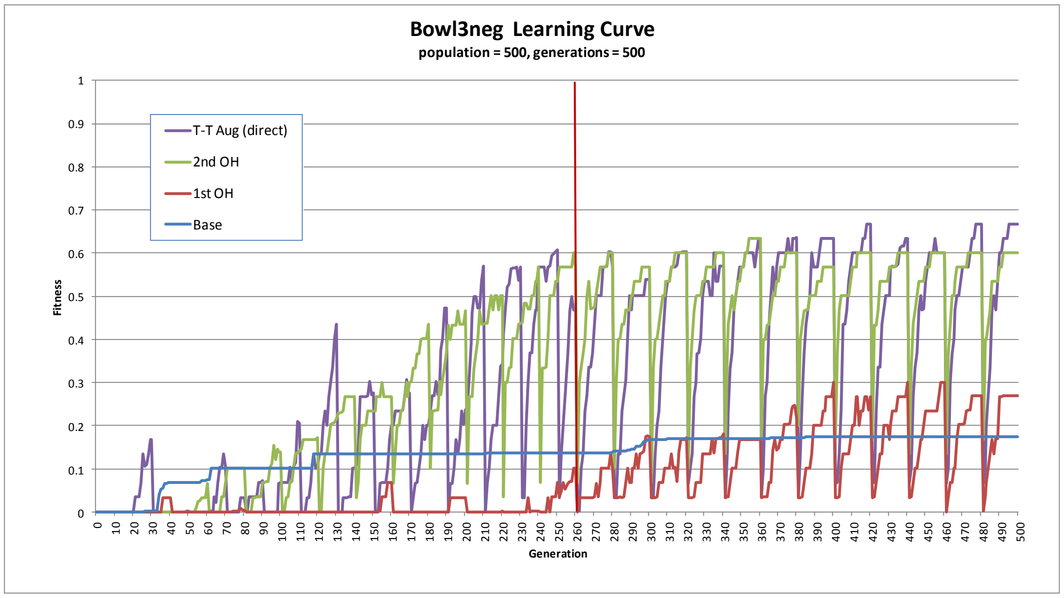

**Figure 15 - *Bowl3neg* Learning Curve versus a Two-tiered Search using a Population of 1000 Seeding a Population of 500 (direct transition)**

Figure 15 shows the result of an ACGP 1st order search with a population of 1000 and 250 generation followed by an ACGP 2nd order search with a population of 500 and 250 generations superimposed on the learning curves shown in Figure 13 with the transition between the 1st tier and the 2nd tier indicated by the red line in the chart. The feed of 1st order ACGP heuristics into the 2nd order ACGP runs used a direct one-to-one transition scheme. The output heuristics of each 1st order ACGP run was fed to a corresponding 2nd order ACGP run. This result is only a marginal improvement over the original experiment from Figure 13. That outcome is not a surprise and it is an artifact of the direct feed of 1st order heuristics into the 2nd order ACGP runs. The direct feed of 1st order heuristics meant that every run was used in this scheme irrespective of the quality of its search. Bad 1st order ACGP runs passed bad heuristics to the second tier 2nd ACGP runs and produced low quality results. Clearly, while this is a simple scheme, it is a suboptimal approach.

The simple direct feed scheme used in the experiment shown in Figure 15 can be modified to improve the results of this methodology. An ideal concept might suggest using only the best 1st order heuristics to precondition the 2nd order ACGP search. This scheme may have merit but introduces the problem of defining what a quality 1st order heuristic solution looks like for a given problem. A simpler method may be good enough to demonstrate the promise of a tiered ACGP search scheme. Figure 16 shows the learning for a tiered search scheme where the 1st order ACGP heuristics are combined into an input for a 2nd order ACGP search by averaging 1st order heuristics from 30 independent runs. The results of this combined transition tiered scheme are shown superimposed over the results from Figure 13. The first stage search uses 1st order ACGP for the initial 250 generations. The second stage uses a conditioned 2nd order ACGP search for the final 250 generations.

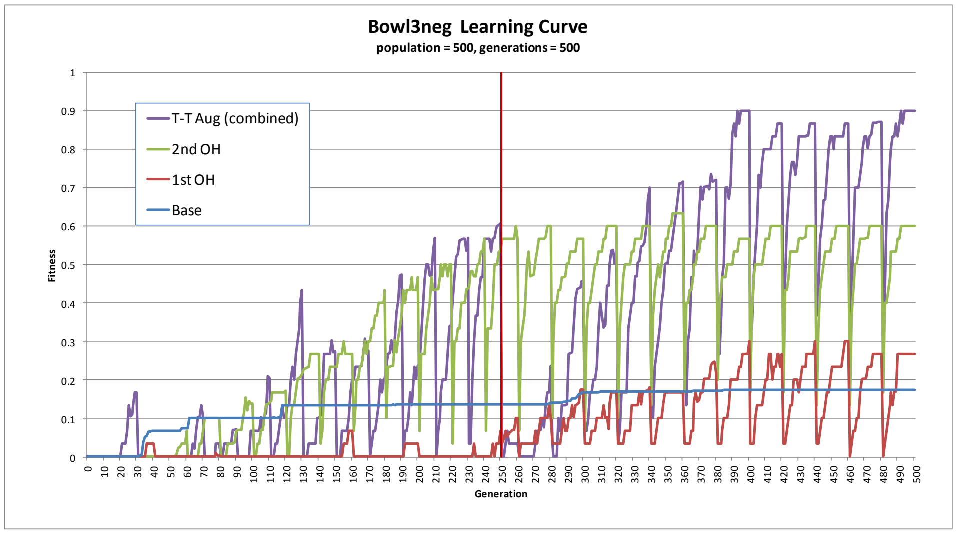

**Figure 16 - *Bowl3neg* Learning Curve using a Population of 500 versus a Two-tiered Search using a Population of 1000 Seeding a Population of 500 (combined transition)**

Figure 17 shows the results of this tiered scheme using a combined 1st order ACGP heuristic superimposed over the results from Figure 14. Again, the first stage search uses 1st order ACGP for the initial 250 generations. The second stage uses a conditioned 2nd order ACGP search for the final 250 generations.

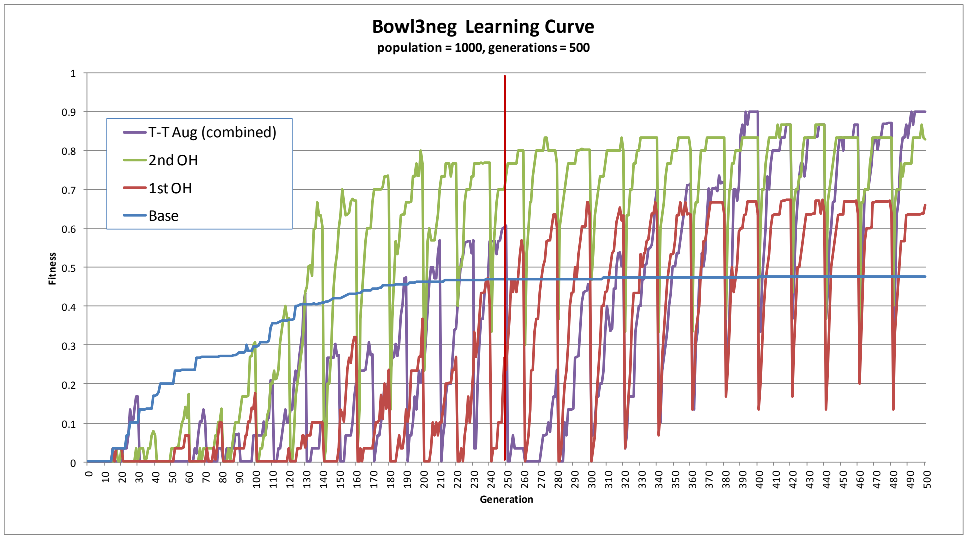

**Figure 17 - *Bowl3neg* Learning Curve using a Population of 1000 versus a Two-tiered Search using a Population of 1000 Seeding a Population of 500 (combined transition)**

The outcome of a combined transition tiered search scheme using the combined average results of 1st order ACGP with a population of 1000 to precondition a 2nd order ACGP search with a population of 500 is effective when compared to the GP and ACGP searches using a population of 500 (Figure 16). The results of Figure 17 do not appear to be as strong. However, in practice, the cost of a method is not the number of generations used but the time it takes. If the learning curves are compared on a time scale rather than on a generational scale, the effectiveness of these results becomes more distinct.

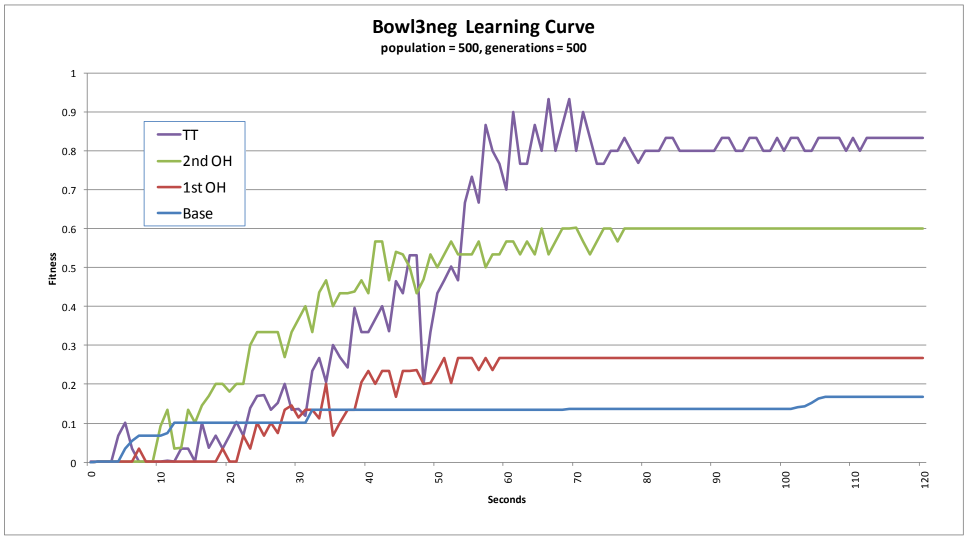

**Figure 18 - Bowl3neg Learning Curve using a Population of 500 versus a Two-tiered Search using a Population of 1000 Seeding a Population of 500(combined transition) compared on a Time Scale**

Figure 18 shows the learning curves from the experiments shown in Figure 16 with the average fitness scores per generation converted to average fitness scores per second (120 second time window). The fitness scores for each run are converted into a continuous time scale with one score per second. Once the time-based fitness scores are processed for each run, they are then averaged to produce the average fitness per second for a method (base GP for example). Since the different methods (base GP, 1st order ACGP, 2nd order ACGP, Two-tier) finish at different time points. The maximum time interval was set to match the average Two-tier execution time. Any method that finished sooner (1st order ACGP using smaller population for example) just has its last run score extended out to the time limit before the run scores are averaged. Any method that ran slower is truncated. This process produces sets of average fitness scores on the same time scale for each set of experiments and creates a chart that demonstrates what average fitness score each technique achieves in each time interval. The advantage of the two-tier scheme, in terms of fitness and execution time, is clear.

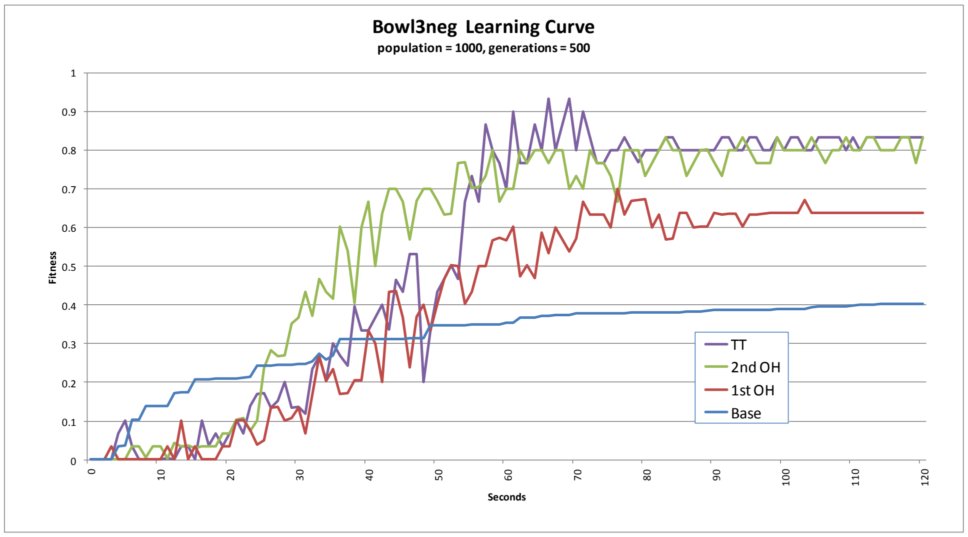

**Figure 19 - Bowl3neg Learning Curve for a Population of 1000 versus a Two-tiered Search using a Population of 1000 Seeding a Population of 500 (combined transition) compared on a Time Scale**

Figure 19 shows the learning curves from the experiments shown in Figure 17 substituting an x-axis representing time (120 seconds) for one representing the training generations. The advantage of the two-tier scheme, in terms of fitness and execution time, is more distinct. The significance of this advantage will be quantified below using statistical testing.

Table 6 presents the average execution times for all of the searches shown in Figure 16 and Figure 17. The average execution times for both of the tiered search schemes described above are comparable with normal ACGP execution times. The smaller search population of 500 individuals executes faster than the larger population but pays a penalty of lower average fitness for the base GP application, 1st order ACGP, and 2nd order ACGP. The Two-tiered approach is faster than both base GP population sizes. Its execution speed is falls between the two population sizes for both versions of ACGP. While these values appear to be numerically distinct, do they represent statistically different sample outcomes?

**Table 6 - Comparison of Average Execution Times (*Bowl3neg*)**

| Execution Times | Population Size | | |
|---|---|---|---|
| | 500 | 1000 | 1000 + 500 |
| Base GP | 272.83 | 411.77 | - |
| ACGP 1st OH | 46.77 | 88.28 | - |
| ACGP 2nd OH | 68.90 | 113.83 | - |
| TT Aug (direct) | - | - | 77.83 |
| TT Aug (combined) | - | - | 77.83 |

Table 7 represents the significance test values for comparisons of the execution times for the set of 30 independent runs of the Two-tiered ACGP methodology versus the sets of 30 independent runs of each of the other six GP searches shown in Table 6. These significance tests sample the set of execution times for each run. Each test compares 30 execution times (one value for each run) for an experiment (base GP for example) versus 30 execution times for the two-tier scheme. The times in each sample are the final value for that run. This process analyzes each run's contribution to the average execution time for a given technique (base GP for example) within the distribution of execution times for that technique. The significance tests compare each pair of samples and determine whether any difference in their distribution is due to normal variance or due to an actual difference in the distributions.

The table shows the p-values for the Mann–Whitney *U* test for each of these comparisons. The Mann–Whitney *U* test is used for these tests because it is a more robust statistic when attempting whether one distribution is stochastically greater than another and does not require normally distributed samples. The *U* test scores are converted to p-values which represent the null hypothesis probability that the two samples differ from each other based solely on chance. A low p-value ($p < 5.0 \times 10^{-2}$) indicates that the two samples are significantly different and that difference is not due to sample variance. Based on the values in Table 7, the differences of the timing averages shown in Table 6 are significant.

**Table 7 - *Bowl3neg* Execution Time Sample Significance (p-values, vs. the Two-tiered method)**

| Execution Times | | | |
|---|---|---|---|
| | Base GP | ACGP 1st OH | ACGP 2nd OH |
| 500 | $1.097 \times 10^{-6}$ | $4.860 \times 10^{-11}$ | $8.179 \times 10^{-4}$ |
| 1000 | $2.072 \times 10^{-2}$ | $1.705 \times 10^{-3}$ | $1.470 \times 10^{-8}$ |

Table 8 shows that the average fitness scores for both of the tiered search schemes described above are an improvement over the base GP and normal ACGP average fitness scores for both population sizes. The combined transition tiered search scheme shows a significant improvement of the average fitness score incurring only a modest cost in average execution time for that experiment.

**Table 8 - Comparison of Average Fitness Scores (*Bowl3neg*)**

| Fitness Scores | Population Size | | |
|---|---|---|---|
| | 500 | 1000 | 1000 + 500 |
| Base GP | 0.1752 | 0.4760 | - |
| ACGP 1st OH | 0.2685 | 0.6615 | - |
| ACGP 2nd OH | 0.6003 | 0.8282 | - |
| TT Aug (direct) | - | - | 0.6668 |
| TT Aug (combined) | - | - | 0.9000 |

Table 9 represents the values of significance tests for comparisons of the fitness scores for the set of 30 independent runs of the Two-tiered ACGP methodology versus the sets of 30 independent runs of each of the other six GP searches shown in Table 8. The table shows the p-values for the Mann–Whitney *U* test for each of these comparisons. Based on the values in Table 9, the differences of the timing averages shown in Table 8 are significant.

**Table 9 - *Bowl3neg* Fitness Score Sample Significance (p-values, vs. the Two-tiered method)**

| Fitness Scores | | | |
|---|---|---|---|
| | Base GP | ACGP 1st OH | ACGP 2nd OH |
| 500 | $4.290 \times 10^{-9}$ | $1.343 \times 10^{-7}$ | $8.701 \times 10^{-4}$ |
| 1000 | $1.002 \times 10^{-5}$ | $1.925 \times 10^{-3}$ | $9.658 \times 10^{-2}$ |

## 5.2 How does this Methodology Reduce the Search Space Complexity?

The tiered search scheme shown above appears to be a promising method to improve ACGP search results for problems with restricted structure quality solutions in the search space. This improvement of the search process is generated by a probabilistic reduction of the components in the initial set of functions and terminals chosen for a given problem. The simple experiment in Chapter 5.1 above demonstrates the efficacy of this proposed methodology. A description of the steps of this methodology is helpful in explaining why it is effective.

Chapter 3.1 conceptually described how 1st order ACGP analyzed the fittest population members and used building block frequency values to adjust their selection probabilities. These probabilities are entered in a table that stores the probability that a specific component will be chosen as an argument for a given function. Each of a function's arguments has its own probability table. These probability tables can be thought of as a variant of the EDA-GP discussed in Chapter 2.3. ACGP however maintains multiple probability tables for each node and only reduces the set of probability tables to a single table for a node when the parent node function is chosen. Figure 20 illustrates how this process works for a component set containing four binary functions. Before the tree root node is populated with a function, the probability tables for all four functions are potentially available at each child node location. The image on the left portrays this situation. Once the root node is assigned a function, as in the image on the right, then only the two tables for each of that function's argument locations remain available for use in selecting the contents of each child node. This behavior is similar to the probability table structure of EDA-GP (Chapter 2.3). The difference between this 1st order ACGP and EDA-GP is that ACGP uses only a single global table. That table assigns a probability to each function and terminal to be selected in a particular argument location for a given function. EDA-GP probability tables evolve location specific probabilities. A given function will be assigned different selection probabilities in different locations of the EDA-GP tree. This location specificity means that an EDA-GP will converge to a specific candidate tree structure and ignore other viable tree structures. The probability tables of 1st order ACGP are not localized. They are global and are applied to all locations in a solution tree. This technique simplifies the computational overhead and encourages the search of diverse candidate tree structures.

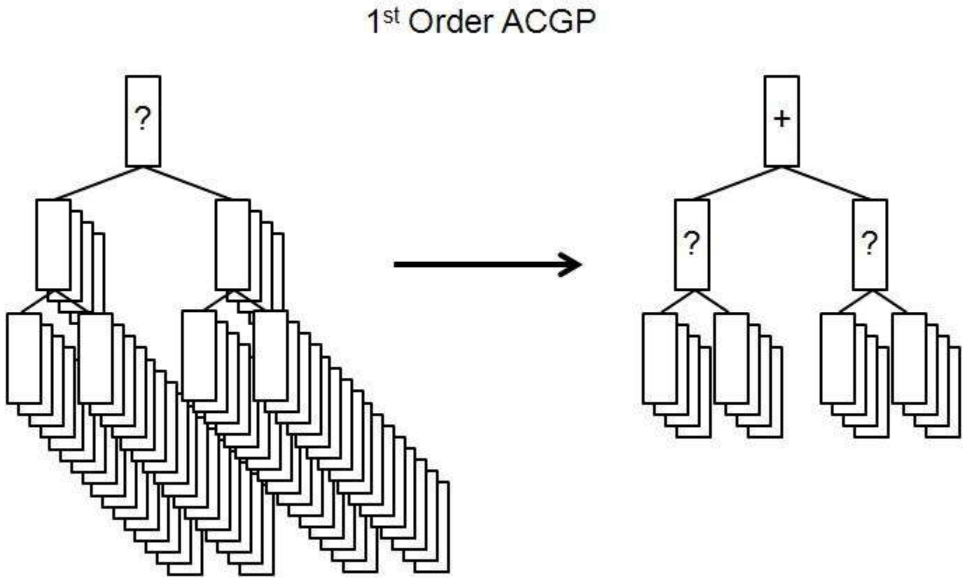

**Figure 20 - 1st order ACGP Heuristic Weights Presented as an EDA Structure**

The 1st order ACGP probabilities can also be logically visualized as a two dimensional table. Table 10 shows an initial unconditioned 1st order ACGP probability table. A function or terminal (the top row) can be assigned to an argument location for a function (two left columns) with a uniform probability of selection (body of the table).

**Table 10 - Initial ACGP 1st Order Heuristic Weight Matrix**

| Func | Arg | * | + | - | / | 0 | 1 | 2 | 3 | 4 | 5 | -1 | -2 | -3 | -4 | -5 | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | 1 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |
| * | 2 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |
| + | 1 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |
| + | 2 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |
| - | 1 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |
| - | 2 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |
| / | 1 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |
| / | 2 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |
| Root | 1 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 | 0.056 |

This table shows the set of functions and terminals used in the *Bowl3neg* (Equation (13)) ACGP search in Chapter 5.1 as the top row of the table. The set of functions and terminals used in this search is:

Function set: $\{+, -, *, /\}$ (protected divide)

Terminal set: {x, y, z, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5}

Each row of Table 10 corresponds to a specific function argument location or the root node. The functions or root are shown in the left column. The argument locations are shown in the second left column. In this example, all functions are binary functions hence the argument designations of 1 or 2. The probabilities of each row represent the selection probabilities for a child node of a given function argument; therefore the sum of each row is 1. An initial table, like this one, will begin with uniform selection probability for all components. This initial behavior is exactly like a standard GP application. The difference between a standard GP implementation and ACGP is that ACGP periodically interrupts normal operation, analyzes the frequency of the building blocks that make up fit population members, and adjusts their selection probability in this table. Those building blocks that occur frequently in fit solutions have their selection probability increased. All other building blocks have their selection probability reduced. Table 11 presents an ACGP 1st order probability table after 250 generations, or 10 iterations, of this adjustment process.

**Table 11 - Example ACGP 1st Order Heuristic Weight Matrix**

| Func | Arg | * | + | - | / | 0 | 1 | 2 | 3 | 4 | 5 | -1 | -2 | -3 | -4 | -5 | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | 1 | 0.047 | 0.134 | 0.133 | 0.127 | 0.021 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.041 | 0.260 | 0.241 |
| * | 2 | 0.024 | 0.136 | 0.210 | 0.073 | 0.013 | 0.020 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.002 | 0.001 | 0.001 | 0.042 | 0.255 | 0.232 |
| + | 1 | 0.151 | 0.115 | 0.207 | 0.041 | 0.074 | 0.014 | 0.009 | 0.024 | 0.048 | 0.039 | 0.028 | 0.045 | 0.011 | 0.020 | 0.008 | 0.103 | 0.061 | 0.015 |
| + | 2 | 0.112 | 0.099 | 0.305 | 0.021 | 0.053 | 0.028 | 0.058 | 0.011 | 0.009 | 0.019 | 0.032 | 0.081 | 0.022 | 0.045 | 0.006 | 0.080 | 0.010 | 0.024 |
| - | 1 | 0.082 | 0.151 | 0.289 | 0.067 | 0.019 | 0.016 | 0.017 | 0.033 | 0.018 | 0.034 | 0.046 | 0.029 | 0.023 | 0.012 | 0.040 | 0.040 | 0.051 | 0.046 |
| - | 2 | 0.285 | 0.145 | 0.125 | 0.019 | 0.012 | 0.023 | 0.001 | 0.051 | 0.035 | 0.021 | 0.034 | 0.047 | 0.031 | 0.029 | 0.044 | 0.050 | 0.034 | 0.029 |
| / | 1 | 0.028 | 0.071 | 0.207 | 0.028 | 0.209 | 0.002 | 0.005 | 0.009 | 0.021 | 0.001 | 0.038 | 0.032 | 0.014 | 0.061 | 0.027 | 0.175 | 0.071 | 0.016 |
| / | 2 | 0.092 | 0.073 | 0.215 | 0.015 | 0.022 | 0.109 | 0.016 | 0.019 | 0.027 | 0.033 | 0.014 | 0.152 | 0.070 | 0.033 | 0.033 | 0.044 | 0.001 | 0.044 |
| Root | 1 | 0.001 | 0.424 | 0.577 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |

The discussion in Chapter 5.1 identified the set of desirable 1st order building blocks in Equation (14). The probabilities for the '–' heuristics appear in rows six and seven. The '*' probabilities are in rows one and two. The columns of the table identify the function or terminal used in a particular function argument node.

Several observations about Table 11 stand out. Initially this table was populated with uniform selection probabilities for four binary functions and fourteen terminals as shown in Table 10. After 250 heuristic adjustment iterations the probabilities are clearly not uniform. This matrix probabilistically eliminates nine terminals and one function (light purple color). While they are not completely eliminated from the component set, they will be selected so seldom that they can be considered eliminated. This probabilistic suppression of the selection of functions and terminals effectively reduces the representation search space derived from Equation (1).

$$\{-_1 *\}, \{-_2 -\}, \{*_1\, y\}, \{*_2\, y\}, \{*_1\, z\}, \{*_2\, z\} \qquad (17)$$

Several of the desirable 1st order building blocks in Equation (17) have increased probabilities (shown in dark green). In this case '–' is a desirable Root node function and it also has an enhanced selection probability. While $\{-_2 *\}$ and $\{-_1 -\}$ are desirable, they are permutations and their lower selection probability is helpful. These are positive observations from this table but there are also some negative aspects in this set of heuristics.

Two of the desirable 1st order building blocks, $\{*_1\, x\}$ and $\{*_2\, x\}$, needed to solve this regression problem had their selection probability reduced from the original uniform value (shown in blue). While these probabilities are lower when compared to their starting values (0.041 and 0.042 versus 0.056), they are considerably greater than most of the highly suppressed values (e.g. $\{*_1\ 1\} = 0.001$) so this is not a major concern and can be remedied in the 2nd order ACGP search. Several unneeded building blocks show increased selection probabilities. The Root function selection of '+' and the heuristic $\{/_1 0\}$ are assigned strong selection probabilities. All of the other building blocks highlighted in light green in the table have enhanced probabilities. While these observations are not optimal, they are not critical. A 1st order probability matrix like this one is rough guidance that can helpfully seed a 2nd order ACGP search.

The 1st order probability matrix in Table 11 is the combined weight matrix used to precondition the 2nd order ACGP search of *Bowl3neg* ACGP search in Chapter 5.1 above. The modified selection probabilities in this table provide coarse guidance for the 2nd order ACGP search that produced the results of Figure 16 and Figure 17 .

An advantage of using this coarse guidance, rather than more refined guidance, is that while the second stage search is constrained, it is only probabilistically constrained. The matrix helps the search with most of the best building blocks yet does not prevent the discovery of other useful elements. Additionally, the probabilistic seeding guidance assists the virtual search locality of productive building blocks and suppresses less productive ones. This behavior assists the search and improves its efficiency. The experiment in Chapter 5.1 demonstrated the efficacy of this method and a formal statement of its parameter choices should help clarify the complete concept.

## 5.3 The Structure of a Two-Tiered ACGP Search Methodology

The results of the two-tiered search scheme presented in Chapter 5.2 above are promising. A discussion of the parameter settings used in this method is helpful in understanding its advantages over unconditioned 1st order or 2nd order ACGP search.

The standard GP searches (base, 1st order ACGP, and 2nd order ACGP) in these experiments with the *Bowl3neg* problem (Equation (13)) used the same operational parameters identified earlier in Table 4. The parameter exceptions for these experiments are the use of populations of 500 and 1000 individuals for the base GP application, 1st order ACGP and 2nd order ACGP. The population sizes for these baseline experiments match the first and second tier populations of the two-tier approach. The fitness and execution time results of these comparison experiments can be compared directly with those of the two-tiered search scheme.

The principal difference in parameters for the two-tiered ACGP search is that this method uses a population of 1000 individuals for 250 generations in 1st order ACGP search. The output 1st order heuristic matrix is then seeded to a 2nd order ACGP search using a population of 500 individuals for 250 generations. These parameters result in strong execution time and fitness results but it may not be obvious why these parameters (population sizes and generation limits) are chosen.

The rationale starts with the question, what constitutes a useable 1st order ACGP heuristic matrix that can seed our second tier run? Ideally, one would prefer a well specified heuristic matrix that only includes the minimal desired components and suppresses everything else. This type of ideal heuristic matrix would provide ideal guidance for the second tier run and help it find a highly fit solution quickly. The cost, in population size and number of generations, of discovering such an ideal heuristic would be high enough to negate any potential advantages for this two-tiered technique. Additionally, if the initial set of 1st order ACGP runs can develop an ideal heuristic matrix, it has probably solved the problem completely thereby making the second stage of this method unnecessary. The concept used here is to evolve a heuristic matrix that is good enough to guide the 2nd order ACGP search and optimize the processing costs. Conceptually, the population must be large enough to provide sufficient diversity yet small enough to retain the original 1st order ACGP processing efficiency.

The experiment in Chapter 5.1 demonstrated that a model developed using a two-tiered search is successful in producing the good results. Additionally, this result provides hints of the environmental parameters for the search. The 1st order ACGP run should use a population large enough to produce several successful searches. The term successful in this case is measured by a search run that results in a best fitness score of 1.0. This group of successful runs does not have to be a majority of the overall set of runs, but they should be more than one or two runs to ensure enough diversity of heuristic information. Empirical results indicate that a set of runs that achieve an average fitness score in excess of 0.5 should contain enough runs with perfect 1.0 fitness scores. The first implementation of this technique tried seeding the 2nd order ACGP runs with a one-to-one set of 1st order ACGP runs. The resulting fitness curve from this direct seeding experiment was shown in Figure 15. Analysis of this experiment revealed that any 1st order ACGP search that was not successful, never produced a successful 2nd order ACGP search. Clearly the poor 1st order heuristics did not contain sufficiently good guidance for the 2nd order searches they seeded. The successful 1st order heuristics were considerably more useful in seeding successful 2nd order searches. This observation indicated that the seed heuristic should exclude any unsuccessful 1st order heuristic from the set of first tier runs. Once the heuristics from the set of runs are pruned from the set of candidate heuristics the question becomes how will the set of candidates be combined into one seed heuristic?

Two desired features of an effective heuristic combination scheme are that it be simple in operation and that it retains the proportionality of the heuristic probabilities within each row of function heuristic probabilities. An arithmetic average of the candidate heuristics proved to be a computationally simple scheme that preserved the probabilistic proportions in each row of the matrix. This combination technique additionally moderates extremely strong or extremely weak probabilities thereby preventing over aggressive convergence to suboptimal structures and thereby suboptimal solutions.

This candidate evaluation process can be automated to quickly process the results from the first tier runs and combine them to produce a seed heuristic matrix for the set of second tier runs. The steps of this methodology can be described as follows:

1) Run 1st order ACGP with typical parameter settings but with half the number of generations than normal (using multiple independent 1st order ACGP runs)

2) Review the final fitness scores for the set of ACGP runs

   a. If there are not a sufficient number of successful runs to combine into a seed heuristic model, increase the population size and repeat step 1
   b. Else, combine the final 1st order heuristic matrices for the successful runs by computing the arithmetic average of each individual heuristic to produce the seed heuristic matrix

3) Seed a set of 2nd order ACGP runs with the 1st order heuristic computed in step 2b.
4) Analyze the final results from the two-tier search.

Several of the parameters of this technique are expressed in conceptual terms. These parameters include: the population sizes for each tier's search, the number of generations for each tier's search, and what constitutes a sufficient number of successful runs. These parameter settings are clarified in the discussion of the empirical results in each chapter below. These experiments will follow a basic protocol. The experiments for the base GP application, 1st order ACGP, and 2nd order ACGP use 30 independent runs with the results of all runs presented as averages of the 30 runs. The two-tier methodology will also use 30 independent runs for each tier with each tier's results presented as averages of the 30 runs. The only modification of this process is that a pruned subset of the first tier results is combined to form the heuristic seed for the 30 independent runs of the second tier.

While the results of the two-tier search in Figure 16 and Figure 17 are apparent, it is not clear whether the 2nd tier fitness is attributable to the search bias induced by the 1st tier search results or the increased information of the conditioned 2nd tier search. One method of resolving this question is to compare the two-tier search results shown in these figures to a two-tier search using 1st order ACGP in both tiers.

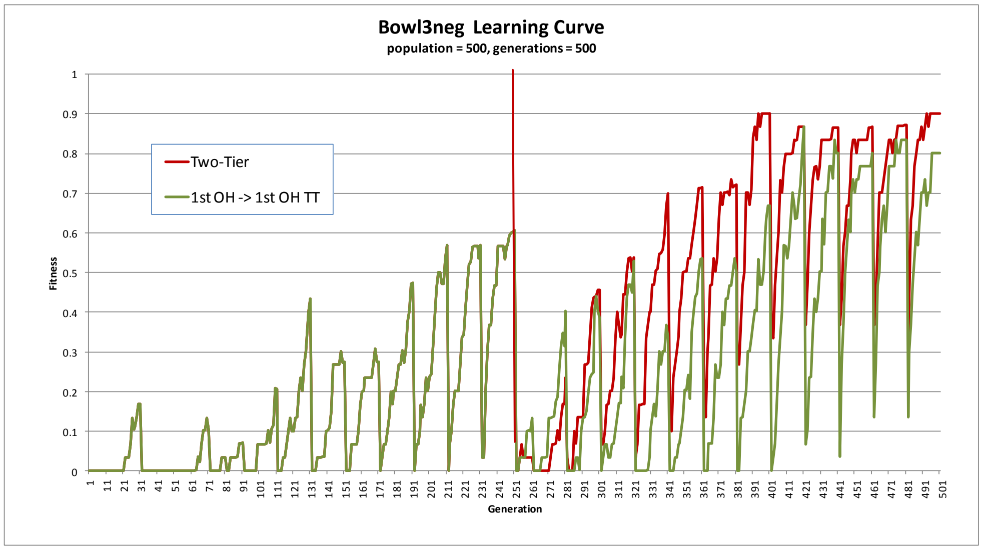

**Figure 21 – *Bowl3neg* Two-Tier Search (1st OH to 2nd OH) versus Two-Tier Search (1st OH to 1st OH)**

Figure 21 compares the two-tier search using an initial 1st order ACGP search that conditions a 2nd order ACGP search to another using 1st order ACGP search that conditions a 1st order ACGP search. Using 2nd order ACGP in the second tier of the tiered search scheme accrues additional information from the use of 2nd order heuristics and thereby has a better average fitness score. This demonstration validates the operational aspects of this proposed search methodology.

This process is simple, effective and easy to validate. The eighteen components of functions and terminals used in the experiment discussed in Chapter 5.2 define a very large representation search space. This matrix resulting from a 1st order ACGP search probabilistically eliminates nine terminals and one function (shown in light purple color in Table 11). This reduction of the GP component set decreases the size of the representation search space by approximately 25% which computationally explains part of the accuracy and efficiency gains. A review of the representation space complexity described by Equation (1) can clarify the scope of the search space reduction achieved here.

Chapter 1.1 discussed the factors that contribute to the complexity of a GP representation search space. This concept is summarized in Equation (1) repeated here.

$$M(level) = \sum_{i}^{F_i} (M(level-1) + |T|)^{arity_{F_i}} \quad where\ M(0) = |F| \qquad (1)$$

This equation expresses the number of potential solutions in a GP representation space in terms of the number of functions, the number of terminals, the arity of each function, and the maximal depth of the trees. The number of terminals is additive factor for the nodes at each depth. While this contributes to the number of permutation structures in the search space, it is not the most significant factor. The functions, which trigger additional levels to solution trees and the arity of the functions which expand the potential permutation options multiplicatively, are more significant influencers. This recursive function is summed, at each descending level, over all of the functions in the component set.

Eliminating a function from the GP component set has a greater impact on reducing the representation search space complexity than eliminating a terminal. In Equation (1), the quantity of possible permutation options are computed for each possible parent function from the set of functions for a given GP application instance. Eliminating a function from the component set eliminates one of these possible parent node choices and thereby reduces one of the elements in the summation in Equation (1). This reduces the size of the representation search space. If the original function set consists of *n* functions, eliminating one function will remove one element in the summation and thereby shrink representation search space by *1/n*.

If a function that is not needed to form a viable solution is eliminated from the original function set, it can cause a significant reduction in the overall complexity of the representation space without reducing the number of viable candidate solutions in this search space. Reducing the overall size of the search space without reducing the number of viable solutions increases the probability that the GP search will discover and exploit a viable solution. This should increase the efficiency of the GP search both in terms of execution time and fitness score. An experiment using an ideal seed heuristic should help reinforce this assertion.

## 5.4 An Experiment with an Ideal Search Heuristic

The experiment in Chapter 5.1 used a 1st order ACGP heuristic matrix as a seed for a 2nd order ACGP search to solve a designed regression problem. The seed matrix was an average of the heuristic matrices from several successful 1st order ACGP searches. The resulting 2nd order ACGP search improved in both time and fitness over standard base GP, standard 1st order ACGP, and standard 2nd order ACGP searches. Chapter 5.2 described the mechanics of this methodology and asserted that, while the 1st order seed matrix was not an ideal guide, it was a sufficient recommendation for a 2nd order ACGP search.

One advantage of using a designed problem is that the target equation is known and, as in this case, its building blocks are also clearly known. While these points can help analyze the search progress and its results, they can also be used to construct an ideal 1st order heuristic matrix as a seed for a 2nd order ACGP search experiment. The *Bowl3neg* symbolic regression problem (Equation (13)) will be used for this exercise. This experiment should demonstrate an upper limit of an ideal case search using the two-tiered methodology discussed here.

Before constructing an ideal 1st order heuristic matrix, a review of the structural constraints for the *Bowl3neg* problem will be helpful. Each row of the heuristic table corresponds to a specific function argument location or the root node. The function and terminal set used in this regression problem are:

Function set: $\{+, -, *, /\}$ (protected divide)

Terminal set: {x, y, z, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5}

The functions used for this regression problem are all binary functions. Fourteen terminals are used in the terminal set. There are ten desirable 1st order building blocks (shown previously in Equation 14). Figure 12 represented an example of a viable solution tree. The frequencies of each of the desirable building blocks in this figure can be used to infer what their un-normalized proportional weights should be in an ideal heuristic matrix. All of the desirable '*' building blocks should have equal weights. The '–' building blocks with '*' as its child node should be weighted twice the value of '–' with a child node of '–' . Finally, each row in the table must sum to approximately 1.0. This information produces this ideal 1st order heuristic weight matrix for Equation (13).

**Table 12 - Example Ideal ACGP 1st Order Heuristic Weight Matrix**

| Func | Arg | * | + | - | / | 0 | 1 | 2 | 3 | 4 | 5 | -1 | 2 | -3 | -4 | -5 | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | 1 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.333 | 0.333 | 0.333 |
| * | 2 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.333 | 0.333 | 0.333 |
| + | 1 | 0.001 | 0.001 | 0.001 | 0.001 | 0.999 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| + | 2 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.999 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| - | 1 | 0.666 | 0.001 | 0.332 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| - | 2 | 0.666 | 0.001 | 0.332 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| / | 1 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.999 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| / | 2 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.999 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Root | 1 | 0.001 | 0.001 | 0.999 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |

A review of Table 12 results in several interesting observations. This table probabilistically eliminates eleven terminals and two functions. Since each row of probabilities must sum to about 1.0, four dummy values are entered in the heuristic matrix: $\{+_1, 0\} = 0.999$, $\{+_2, 1\} = 0.999$, $\{/_1, 2\} = 0.999$, and $\{/_2, 3\} = 0.999$ (shown in blue). These probabilities will be rarely chosen because their two parent functions ('+' and '/') are probabilistically eliminated from consideration. Based on the discussion from Chapter 5.3, it is clear that this table probabilistically reduces the size of the representation search space by 50%. That is a significant impact.

Feeding this 1st order seed matrix to a 2nd order ACGP search produces the following average fitness result. This experiment used the GP parameters listed in Table 4.

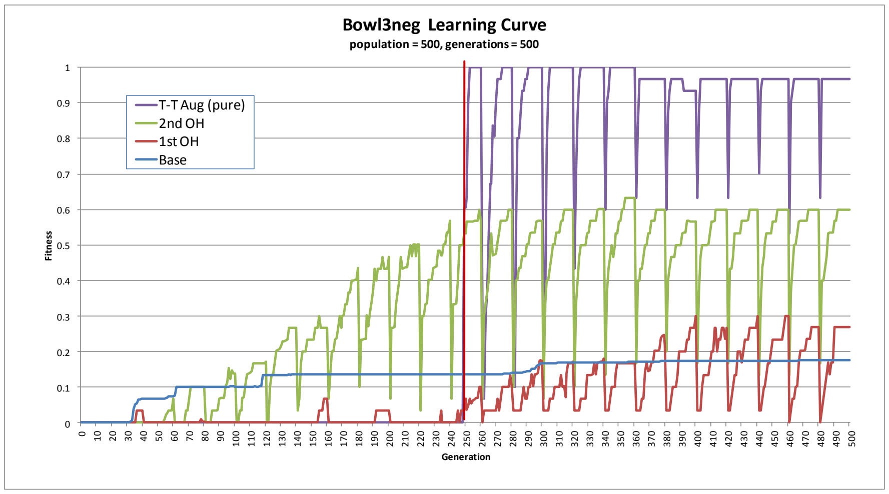

**Figure 22 - ACGP 2nd Order Search Using an Ideal Heuristic versus Normal ACGP Search Using a Population of 500 Individuals**

The results shown in Figure 22 are dramatic. The positive impact of seeding a 2nd order ACGP search with the ideal 1st order ACGP heuristic matrix is clear. Additionally, behavior of a probabilistic system is also visible in this chart. Even though this search was preconditioned using an ideal 1st order heuristic matrix, one of the 30 independent runs lost its solution. This is visible in Figure 22 following generation 392 and persisting to the end of that run. This behavior is typical of a probabilistic heuristic search and does not detract from the clear advantage of this methodology.

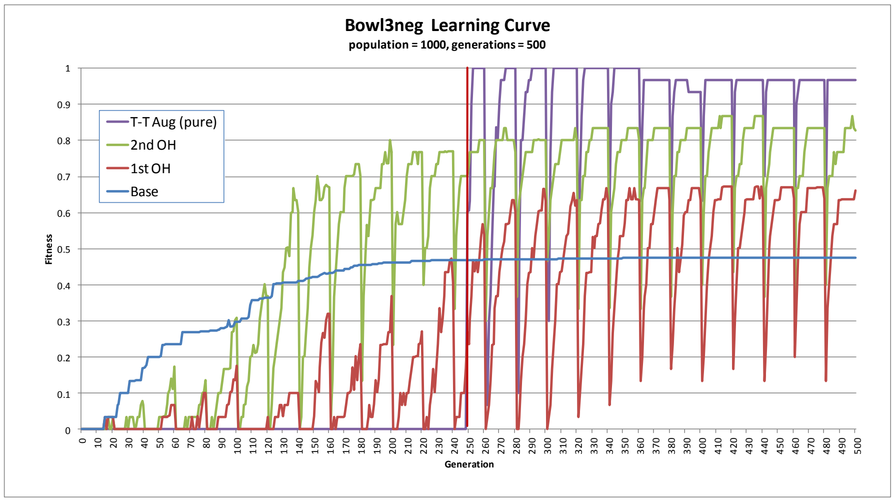

**Figure 23 - ACGP 2nd Order Search Using an Ideal Heuristic versus Normal ACGP Search Using a Population of 1000 Individuals**

Figure 22 compares the ideal two-tiered methodology results to a standard base GP search, a standard 1st order ACGP search, and a standard 2nd order ACGP search using a population of 500 individuals. Figure 23 compares the same ideal two-tiered methodology results to a standard base GP search, a standard 1st order ACGP search, and a standard 2nd order ACGP search using a population of 1000 individuals. These results are still a strong improvement over the other searches.

Both of these experiments demonstrate that 2nd order ACGP can find a correct solution within a very few generations if seeded with an ideal 1st order ACGP heuristic matrix. Table 13 shows the strong execution time advantage when a perfect heuristic is used. This is not a fair comparison though because the experiment using the perfect heuristic did not need the initial run of 250 generations to generate the seed heuristic used in the seeded 2nd order ACGP run, therefore the time value for this experiment reflects only the 250 generations using the perfect heuristic seed.

**Table 13 - Comparison of Average Execution Times (*Bowl3neg* - with Perfect heuristic)**

| Execution Times | Population Size | | |
|---|---|---|---|
| | 500 | 1000 | 1000 + 500 |
| Base GP | 272.83 | 411.77 | - |
| ACGP 1st OH | 46.77 | 88.28 | - |
| ACGP 2nd OH | 68.90 | 113.83 | - |
| TT Aug (combined) | - | - | 77.83 |
| TT Aug (perfect) | - | - | 27.07 |

The significance tests for the perfect heuristic seed experiment sample (Table 14) are clear evidence that this test produced a strong result versus the experiment samples for all of the other experiments. As previously stated, the Mann–Whitney *U* test is used for these tests because it is a more robust statistic when attempting whether one distribution is stochastically greater than another.

**Table 14 - *Bowl3neg* Execution Time Sample Significance versus Perfect Heuristic (p-values)**

| Execution Times | | | |
|---|---|---|---|
| | Base GP | ACGP 1st OH | ACGP 2nd OH |
| 500 | $1.698 \times 10^{-7}$ | $2.285 \times 10^{-10}$ | $2.894 \times 10^{-11}$ |
| 1000 | $3.834 \times 10^{-4}$ | $2.916 \times 10^{-11}$ | $2.930 \times 10^{-11}$ |

Table 15 shows the strong advantage in fitness score produced when a perfect heuristic is used. While the experiment using the perfect heuristic did not need the initial run of 250 generations to generate the seed heuristic used in the seeded 2nd order ACGP run, this result, unlike the execution time result, is a fair comparison because in each the best average fitness scores are directly comparable. These results are not dependent on the number of generations necessary to produce them.

**Table 15 - Comparison of Average Fitness Scores (*Bowl3neg* - with Perfect heuristic)**

| Fitness Scores | Population Size | | |
|---|---|---|---|
| | 500 | 1000 | 1000 + 500 |
| Base GP | 0.1752 | 0.4760 | - |
| ACGP 1st OH | 0.2685 | 0.6615 | - |
| ACGP 2nd OH | 0.6003 | 0.8282 | - |
| TT Aug (combined) | - | - | 0.9000 |
| TT Aug (perfect) | - | - | 1.0000 |

The Mann–Whitney significance test results (Table 16) for the fitness score sample of the perfect heuristic experiment versus samples of each of the other experiments demonstrate a unmistakable advantage in using the perfect 1st order ACGP heuristic as the seed for a 2nd order ACGP run. While this exercise helps understand the advantage promised by this two-tier search methodology, a perfect 1st order heuristic is not guaranteed and may not be necessary. The discussion in Chapter 5.2 regarding the 1st order ACGP seed heuristic for the *Bowl3neg* problem pointed out that the heuristic seed matrix (Table 11) that created the fitness curves shown in Figure 16 and Figure 17 did not have strong selection probability heuristics for all of the desirable building blocks yet it produced a strong result. One might wonder if this methodology is effective with more complex problems. The *artificial ant* problem is often used as a benchmark for research results and would be a good test for this two-tiered search methodology.

**Table 16 - *Bowl3neg* Fitness Score Sample Significance versus Perfect Heuristic (p-values)**

| Fitness Scores | | | |
|---|---|---|---|
| | Base GP | ACGP 1st OH | ACGP 2nd OH |
| 500 | $1.700 \times 10^{-10}$ | $1.344 \times 10^{-8}$ | $1.304 \times 10^{-4}$ |
| 1000 | $8.051 \times 10^{-7}$ | $2.870 \times 10^{-4}$ | $1.985 \times 10^{-2}$ |

## 5.5 Two-tiered Search Applied to the *Artificial Ant* Problem

The *artificial ant* problem is a well-studied GP problem that dates back to Koza's original work [12]. The artificial ant problem directs a GP application to evolve a program of instructions for a finite-state automaton (the ant). The ant must find and consume all of the food pellets lying along an irregular trail [12]. In the Santa Fe Trail version of this problem the candidate programs are evaluated by having the ant navigate a 32 x 32 grid containing an irregular trail of 89 scattered food pellets.

Figure 24 depicts a diagram of the Santa Fe Trail grid showing the path of food pellets. The ant starts in the upper left corner. Each candidate solution is evaluated by running the program for 400 time units (repeat the program execution 400 times) counting the number of food pellets consumed. The more pellets consumed during the execution, the higher the score with a perfect fitness score awarded for consuming all 89 pellets.

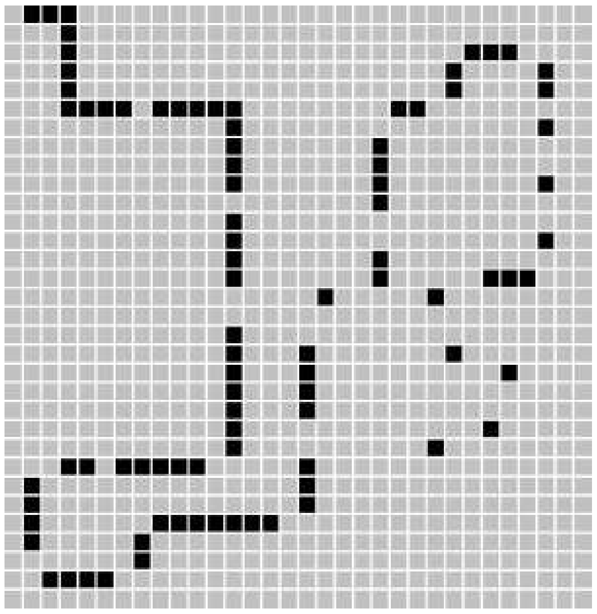

**Figure 24 - The Santa Fe Trail for the *Artificial Ant* Problem**

The set of functions and terminals normally used for this problem are:

Function set:

If-food-ahead (two arguments)

Prog2 (two arguments)

Prog3 (three arguments)

Terminal set:

Right (turns the ant right by 90° without advancing the ant)

Left (turns the ant left by 90° without advancing the ant)

Move (moves the ant forward in the direction it is facing, it eats any food in the square)

These components are combined and manipulated by the GP application to evolve successful programs. It should be noted that the *if-food-ahead* function used in this problem differs from other implementations. Normally a typical GP *if* function has three arguments. The first argument is a Boolean test. Based on the value of that test one of the other two arguments is chosen (argument two for a *true* value and argument three for a *false* value). The *if-food-ahead* function in the function set for the *artificial ant* problem assumes a built-in logic test for *if-food-ahead* and it has only two arguments for its resulting action based on the value of this test. Janikow [9] identified one successful solution to this problem (Equation (18)).

$$\Big(if - food - ahead\ move\ \Big(prog3\ right\ \Big(if - food - ahead\ move\ \big(prog3\ left\ left\ (if - food - ahead\ move\ right)\big)\Big)\ move\Big)\Big) \tag{18}$$

Equation (18) contains five function elements and seven terminal statements. Organizing these logic statements into a tree structure we get the 13 node tree shown in Figure 25. One unmistakable structural feature of this minimal solution tree is its skewed construction. Some quick experimentation with this structure makes it clear that any alteration of this solution will significantly modify its operation and therefore its fitness evaluation. This observation implies that, while other program structures can produce successful solutions to this problem, permutations of the components will probably not produce successful solutions. This behavior is similar to the strict structure created by strict-structure functions addressed in the discussion of the *Bowl3neg* problem. This assertion implies that this problem will have similar strict solution structure behavior as the *Bowl3neg* problem.

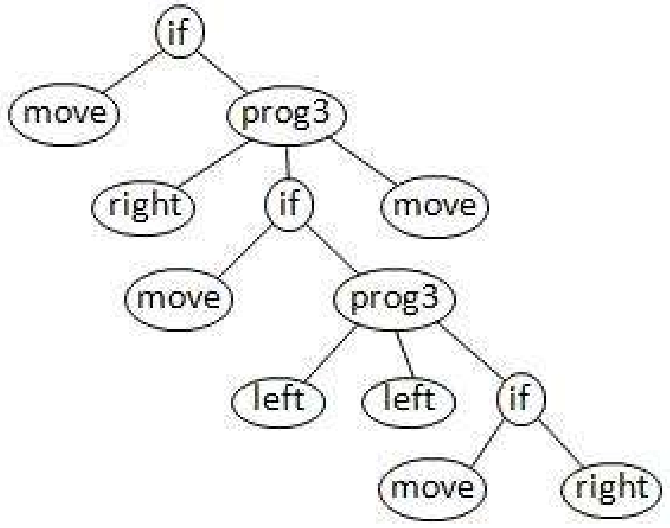

**Figure 25 – An Example Solution Tree for the Artificial Ant Problem (Santa Fe Trail)**

Experimentation with the *artificial ant* problem verified that the strict solution structure created by the functions used in the component set makes it more difficult to find fit individuals within the population of candidate solutions. These experiments indicated that larger population sizes were needed to attain reasonable results. Two sets of 30 independent runs using populations of 1000 and 2000 were made to establish the performance baseline for the *artificial ant* problem. All of the GP searches for the *artificial ant* problem used the operational parameters shown in Table 4, the set of functions and terminals shown above, along with the following exception for this problem:

Population size: 1000 and 2000

The experiments (base GP application, 1st order ACGP, and 2nd order ACGP) using a population of 1000 individuals resulted in the fitness learning curves shown in Figure 26. The base GP implementation was only able to achieve a fitness of 0.09 using this population size. 1st order ACGP and 2nd order ACGP did better but these results were still modest. It is worth noting that 1st order ACGP outperformed 2nd order ACGP on this problem. It is unclear whether any information differential exists between the two ACGP operating modes and this may potentially explain some portion of these results.

What can explain the fact that 1st order ACGP has stronger fitness scores than 2nd order ACGP in Figure 26? One potential explanation may be the difference in search granularity of the two methods with the lower granular 1st order heuristics being able to learn its simpler heuristics quicker. Additionally, the more granular 2nd order heuristics may not provide a considerable information advantage for this problem.

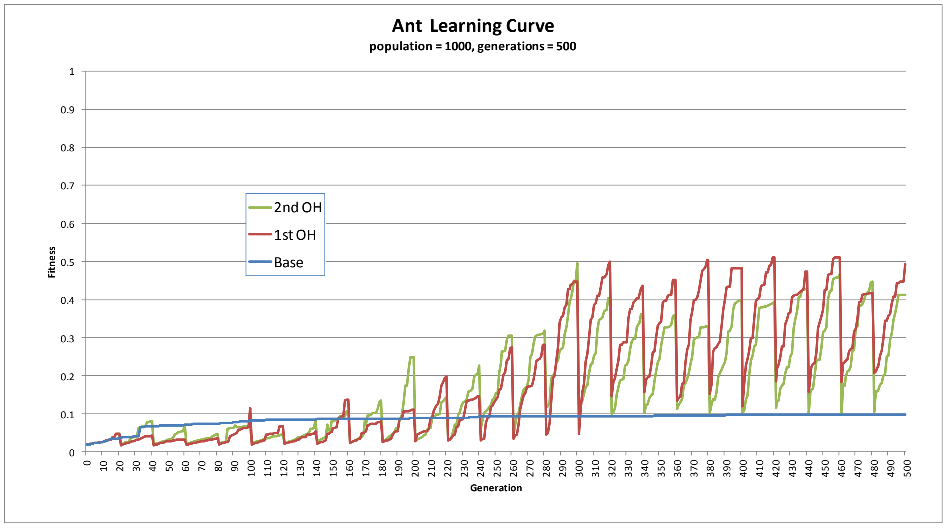

**Figure 26 - *Artificial Ant* Learning Curve (population 1000)**

Similar experiments (base GP application, 1st order ACGP, and 2nd order ACGP) using a population of 2000 individuals resulted in the fitness learning curves shown in Figure 27. The base GP implementation achieved a better fitness score using this population size. 1st order ACGP and 2nd order ACGP also improved their fitness scores. Again, 1st order ACGP and 2nd order ACGP achieved similar fitness on this problem. This fitness result reinforces the assertion that either the difference in search granularity of the two methods or the lack of an informational advantage handicaps the 2nd order ACGP search for this problem environment.

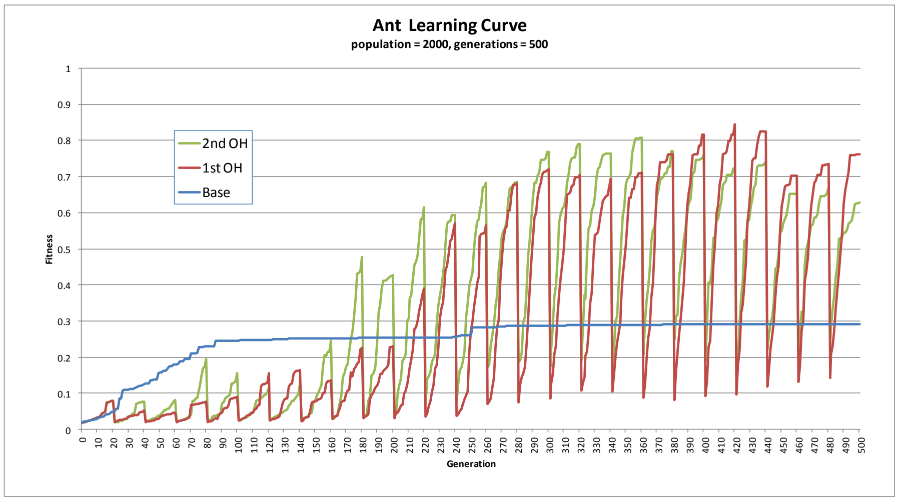

**Figure 27 - *Artificial Ant* Learning Curve (population 2000)**

The two-tiered search methodology was applied to this problem. The first stage consisted of 30 independent runs of 1st order ACGP using a population of 2000 for 250 generations. The results of these runs were analyzed. Any run that was not successful was pruned from the heuristic set. Success was defined by the achievement of a fully fit solution for this problem by the completion of the run. The 1st order heuristics for the successful runs were combined by arithmetic averaging to form the seed heuristic for the second search stage. The second stage consisted of 30 independent runs of 2nd order ACGP seeded with the 1st order heuristic produced by the first stage search and using a population of 2000 for 250 generations. The results of this experiment are shown in Figure 28. This chart shows the two-tiered fitness learning curve superimposed on the set of results for the base GP application, 1st order ACGP, and 2nd order ACGP using a population of 2000. The two-tier search methodology appears to show an improvement over any of the other three GP searches for this problem.

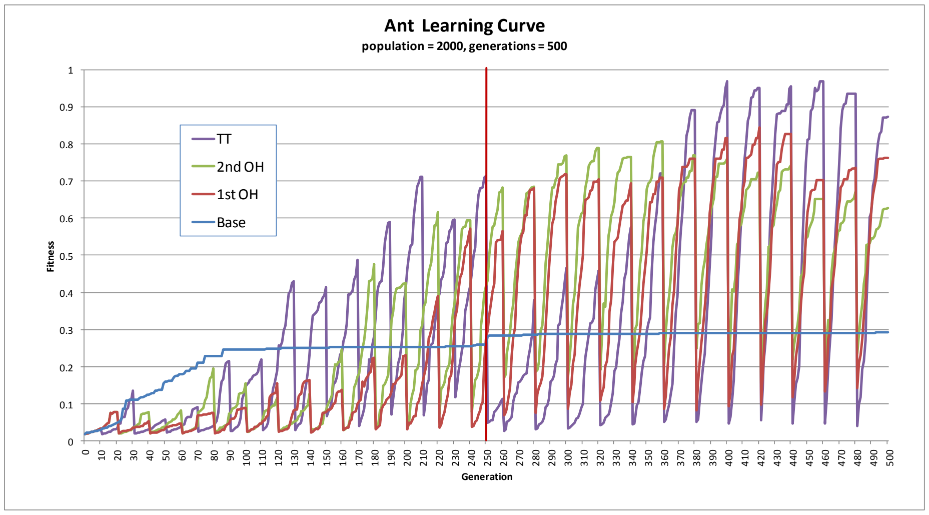

**Figure 28 - *Artificial Ant* Learning Curve using a Population of 2000 versus the Two-Tier Method using a Population of 2000 seeding a Population of 1000**

Figure 29 shows the two-tiered fitness learning curve compared with the set of results for the base GP application, 1st order ACGP, and 2nd order ACGP using a population of 1000. The improvement of the two-tier search methodology over any of the other three GP searches is very strong here.

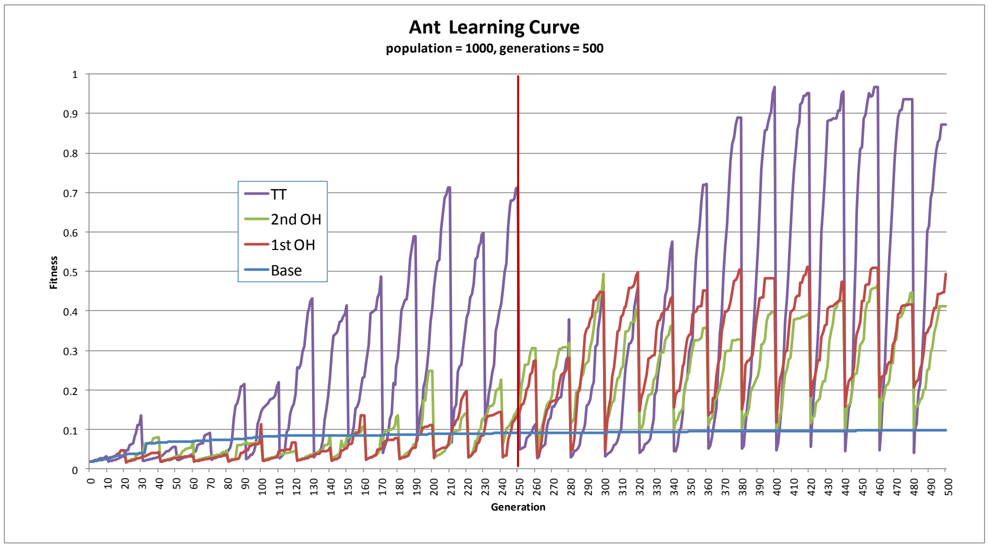

**Figure 29 - *Artificial Ant* Learning Curve using a Population of 1000 versus the Two-Tier Method using a Population of 2000 seeding a Population of 1000**

Earlier in this chapter it was stated that it is unclear whether any information differential exists between the two ACGP operating modes for this problem and that this may potentially explain some portion of the results for the two ACGP modes alone (Figure 26 and Figure 27). While the results of the two-tier scheme shown in Figure 28 and Figure 29 are good, it is not apparent whether these results are attributable to 2nd order ACGP search in the second tier or a just biased 2nd tier search using either ACGP mode. An experiment similar to the one shown in Figure 21 that compares the two-tier search using 1st order ACGP in the 1st tier and 2nd order ACGP in the 2nd tier versus another two-tier search using 1st order ACGP in both tiers may resolve this question. Figure 30 compares the two-tier search results from Figure 29 to a two-tier search using 1st order ACGP in both tiers. The use of 2nd order ACGP in the 2nd tier is better than a 2nd tier search using 1st order ACGP. This reinforces the claim that 2nd order ACGP has an information advantage over 1st order ACGP when the representation space complexity can be constrained.

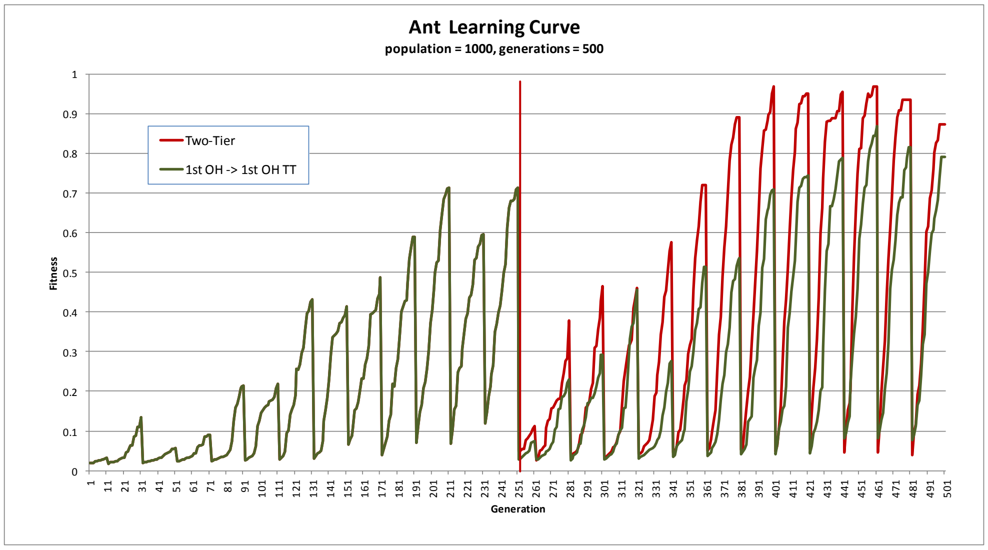

**Figure 30 – *Artificial Ant* Two-Tier Search (1st OH to 2nd OH) versus Two-Tier Search (1st OH to 1st OH)**

The results for the two-tiered search method shown in Figure 28 and Figure 29 appear to indicate that this technique improves the search effectiveness in a combination of search time and overall fitness score. This claim can be verified by reviewing the execution time and fitness score results directly.

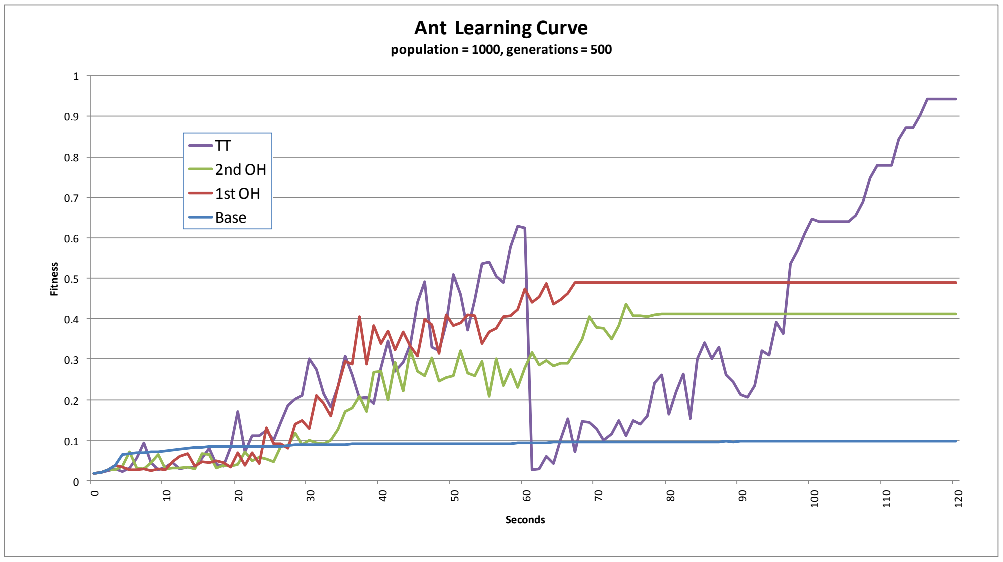

**Figure 31 - *Artificial Ant* Learning Curve using a Population of 1000 versus the Two-Tier Method using a Population of 2000 seeding a Population of 1000 compared on a time scale**

Figure 31 shows the learning curves from the experiments shown in Figure 28 substituting an x-axis representing time (120 seconds) for one representing generations. The advantage of the two-tier scheme, in terms of fitness and execution time, is clear.

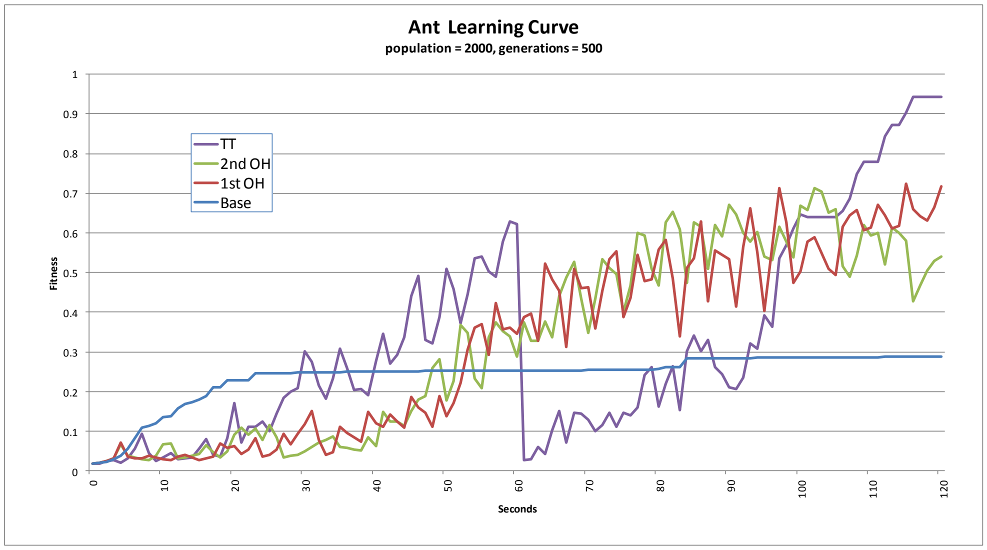

**Figure 32 - *Artificial Ant* Learning Curve using a Population of 2000 versus the Two-Tier Method using a Population of 2000 seeding a Population of 1000 compared on a time scale**

Figure 32 shows the learning curves from the experiments shown in Figure 29 substituting an x-axis representing time (120 seconds) for one representing generations. The advantage of the two-tier scheme, in terms of fitness and execution time, is more distinct. Table 17 compares the average execution time for each set of runs for each search method. The two-tier search is fast and improves on most of the other searches. Only 1st order ACGP and 2nd order ACGP are faster than the two-tier search method.

**Table 17 - Comparison of Average Execution Times (*Artificial Ant*)**

| Execution Times | Population Size | | |
|---|---|---|---|
| | 1000 | 2000 | 2000 + 1000 |
| Base GP | 115.07 | 184.50 | - |
| ACGP 1st OH | 59.23 | 119.33 | - |
| ACGP 2nd OH | 73.73 | 134.27 | - |
| TT Aug (combined) | - | - | 96.70 |

When the samples of execution time values for the set of independent runs for each method are compared with the sample of values for the two-tier method runs (Table 18) it is clear that these timing results are not the result of chance. The two-tier method is only slower than the two ACGP searches with the smaller population. This table shows the p-values for the Mann–Whitney *U* test for each of these comparisons. The Mann–Whitney *U* test is used for these tests because it is a more robust statistic when attempting whether one distribution is stochastically greater than another.

**Table 18 - *Artificial Ant* Execution Time Sample Significance (p-values, vs. the Two-tiered method)**

| Execution Times | | | |
|---|---|---|---|
| | Base GP | ACGP 1st OH | ACGP 2nd OH |
| 1000 | $2.262 \times 10^{-6}$ | $2.618 \times 10^{-11}$ | $4.954 \times 10^{-11}$ |
| 2000 | $3.735 \times 10^{-4}$ | $1.002 \times 10^{-10}$ | $2.899 \times 10^{-11}$ |

Table 19 compares the average fitness scores for each set of runs for each search method on the *artificial ant* problem. Here the two-tier search improves on the fitness scores from all of the other search methods. The strong fitness score of the two-tier search method offsets any execution time advantage 1st order ACGP or 2nd order ACGP had in the Table 17 results.

**Table 19 - Comparison of Average Fitness Scores (*Artificial Ant*)**

| Fitness Scores | Population Size | | |
|---|---|---|---|
| | 1000 | 2000 | 2000 + 1000 |
| Base GP | 0.0981 | 0.2916 | - |
| ACGP 1st OH | 0.4931 | 0.7626 | - |
| ACGP 2nd OH | 0.4131 | 0.6282 | - |
| TT Aug (combined) | - | - | 0.7968 |

When the samples of fitness score values for the set of independent runs for each method are compared with the sample of values for the two-tier method runs it is clear that these fitness results (Table 20) are unmistakable learning improvements and not the result of chance. This table shows the p-values for the Mann–Whitney *U* test for each of these comparisons. The Mann–Whitney *U* test is used for these tests because it is a more robust statistic when attempting whether one distribution is stochastically greater than another.

**Table 20 - *Artificial Ant* Fitness Score Sample Significance (p-values, vs. the Two-tiered method)**

| Fitness Scores | | | |
|---|---|---|---|
| | Base GP | ACGP 1st OH | ACGP 2nd OH |
| 1000 | $4.754 \times 10^{-9}$ | $4.298 \times 10^{-3}$ | $6.729 \times 10^{-4}$ |
| 2000 | $1.707 \times 10^{-5}$ | $6.688 \times 10^{-1}$ | $8.897 \times 10^{-2}$ |

Although viable solutions for the *artificial ant* problem require strict functional structures that do not lend themselves to permutations that are also viable solutions, this two-tier search methodology can discover solutions with better fitness than a standard GP, 1st order ACGP, or 2nd order ACGP techniques and do so more efficiently. An assumption that drives this performance is that the population for the first stage search by 1st order ACGP be large enough so structural diversity of the population ensures candidate solutions that can contribute useful building block information. Without this information, ACGP may bias the search toward suboptimal solutions. An experiment with the *ComplexEq* problem (Equation (9)) introduced in Chapter 3.2 should help explore the question of population size and building block diversity.

## 5.6 Two-tiered Search Applied to a Complex Regression Problem

A specifically constructed regression problem was introduced in Chapter 3.2 as an example experiment demonstrating how ACGP can have difficulties with problems whose viable solutions require strict structure out of a complete population of candidate solutions. The *ComplexEq* problem was defined by Equation (9) and an example solution tree was shown in Figure 9. As discussed in Chapter 3.2, this problem has a slight information advantage for 2nd order ACGP versus 1st order ACGP. The average fitness curves shown in Figure 10 and Figure 11 did not appear to support this observation.

The experimental results in Figure 10 and Figure 11 indicated that without a considerably large population ACGP search is not very successful. In fact, experiments show that, with populations below 2000 individuals, a base GP implementation outperforms the fitness of either ACGP method. Even with larger populations, 1st order ACGP appears to attain better fitness scores than 2nd order ACGP. Two factors conspire to create this behavior. First, the target equation is complex. Its tree depth and component organization implies a requirement for both larger trees and a more diverse population of those trees. Additionally, the inclusion of the strict-structure function '–' imposes a strict structure on viable candidate solutions. These two factors Increase the search complexity for 2nd order ACGP to the point where it becomes inefficient.

Two experiments using large populations should help demonstrate these points. Each experiment used 30 independent searches of either base GP application, 1st order ACGP, or 2nd order ACGP. All of the GP searches for the *ComplexEq* problem used the operational parameters listed in Table 4 with the following exceptions for this problem:

Population size: 5000 and 10,000

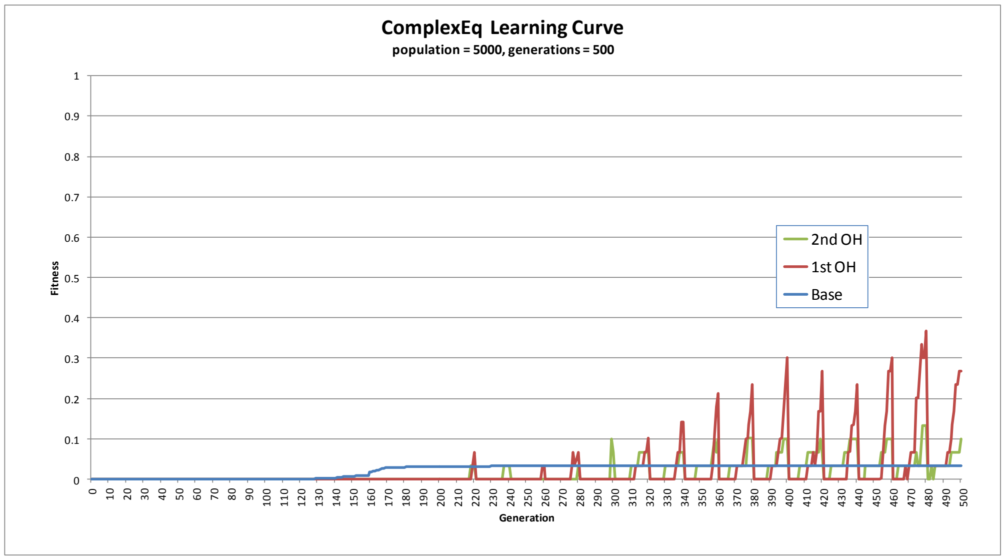

**Figure 33 - *ComplexEq* Average Fitness Curve (population 5000)**

Figure 33 portrays the average fitness results for the three GP methods using a population of 5000 individuals. The base GP application began to show fitness results soonest but its fitness scores also quickly stopped improving. This performance plateau can possibly be attributed to introns and solution bloat [3], [12], [13]. While the two versions of ACGP began improving later than the base GP application, their fitness results were better with 1st order ACGP achieving stronger scores than 2nd order ACGP.

The slower overall ACGP learning rates and the advantage of 1st order ACGP over 2nd order ACGP appear to indicate a learning handicap ACGP has with complex solution structures and problems with strictly structured candidate solutions. The increased number of desirable 2nd order heuristics in this problem (see Equation (12)) means that 2nd order ACGP will take longer to discover all of them and bias its search to use them.

One way this problem can be mitigated is to increase the size of the population thereby increasing the diversity of the population and increasing the probability of finding better candidate solutions in that population.

The fitness results of the three GP searches using a population of 10,000 are shown in Figure 34. The base GP application is the first to show fitness improvement but then stalls because of the growth bloat in population members. This time 2nd order ACGP has quick fitness gains versus 1st order ACGP but eventually 1st order ACGP achieves higher fitness scores. Before progressing on to an experiment with the two-tier search methodology it is worth asking the question, what are the implications of these two experiments and want impacts would they have on a two-tier ACGP search?

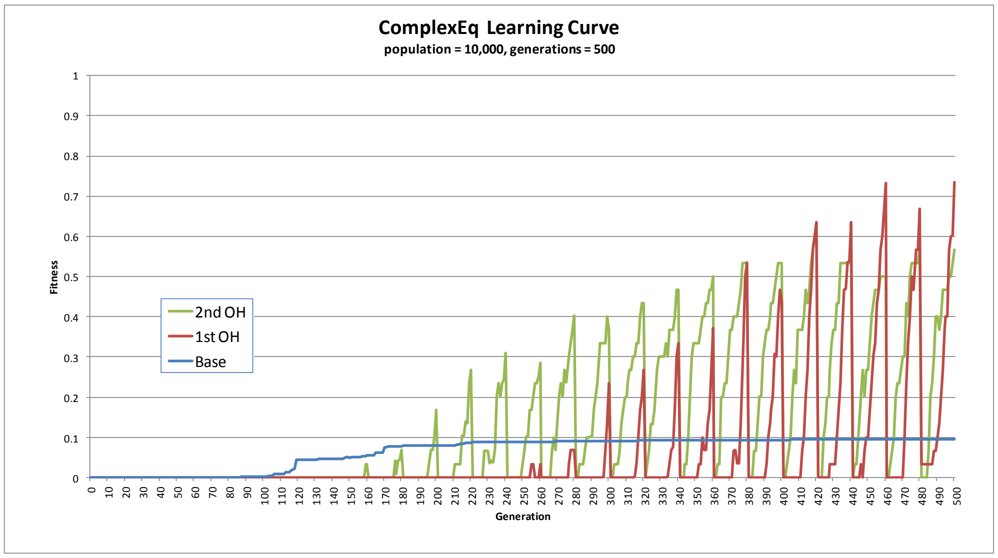

**Figure 34 - *ComplexEq* Average Fitness Curve (population 10,000)**

The first observation that should be noted is that both ACGP methods are slower in developing the average fitness for all 30 independent runs compared to the base GP implementation. ACGP has difficulty discovering desirable building blocks in more complex problems as seen in Figure 8, Figure 14, and Figure 27. In each of these cases, ACGP conducts its early analysis of the building blocks and adjusts their heuristics without enough viable solutions in the population. Once enough viable solutions are available in the population, then ACGP easily discovers the desirable building blocks that contribute to the success of these solutions and enhances the selection probabilities for those building blocks. This observation indicates that the effectiveness of ACGP search is dependent on finding viable solutions in a given population.

The second observation, that 1st order ACGP appears to learn more successfully on this problem than 2nd order ACGP, suggests some implications regarding the complexity of the two ACGP methods. Equation (1) can be used to compute the complete number of 1st order and 2nd order building blocks for a given set of functions and terminals. The set of 1st order building blocks always has fewer elements than the set of 2nd order building blocks. While this is fairly obvious, it has implications for the different complexity of each method's search and how easily an ACGP search might be deceived. Since a 1st order ACGP search analyzes fewer heuristics, it has a higher probability of discovering useful building blocks, even by chance. The more complex 2nd order ACGP search is dependent on the discovery of fit solutions that contain desirable building blocks. It cannot rely on chance to discover them.

An experiment with the two-tiered search methodology for the *ComplexEq* problem will attempt to capitalize on the advantage of 1st order ACGP search to seed a 2nd order ACGP search. This experiment uses the same parameters as the previous experiments with 30 independent runs of 1st order ACGP with a population of 10,000 for 250 generations. The result of each run is reviewed and the 1st order heuristic matrix of any unsuccessful run is removed from the set of heuristic matrices. A successful run is defined as one that results in a fitness score of 1.0. The remaining 1st order heuristic matrices are combined to form the seed heuristic. This seed is used to condition a 2nd order ACGP search with a population of 5000 for 250 generations.

Figure 35 shows the results of this experiment versus the other experimental results for a population of 5000 (Figure 33). The larger population of the 1st order ACGP search (10,000 individuals) produces in a stronger fitness result in the initial 250 generation stage of the two-tier search. The 1st order seed heuristic, produced by the first stage search, conditions the 2nd order ACGP search and initializes that stage of the two-tier search method with a population of more viable solutions. The second stage of the two-tier search quickly discovers useful heuristics and enhances their use even though this stage uses a smaller (5000 individuals) population compared to the first stage.

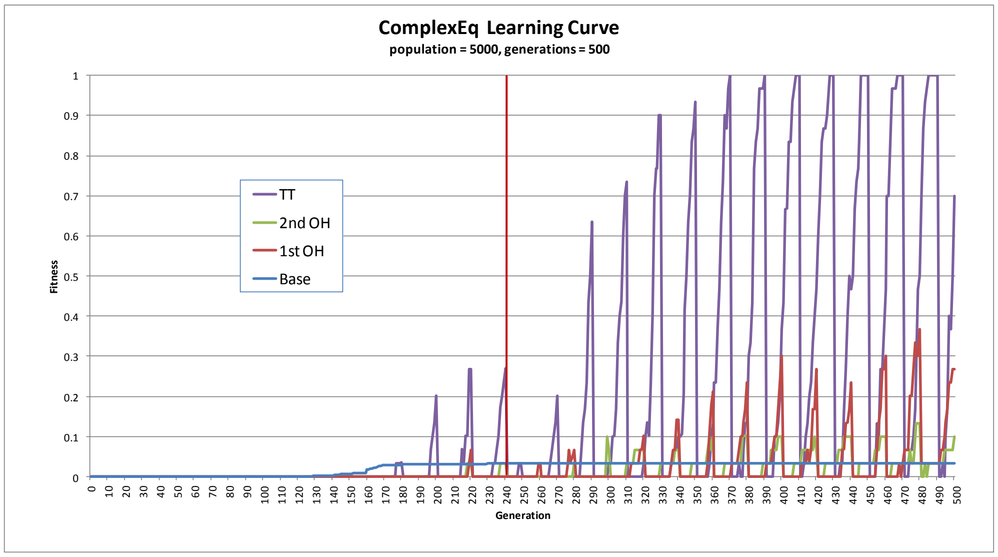

**Figure 35 - *ComplexEq* Average Fitness Curve using a Population of 5000 versus Two-Tier Scheme using a Population of 10,000 seeding a Population of 5000**

Figure 36 shows the results of this experiment versus the other experimental results for a population of 5000 (Figure 34).

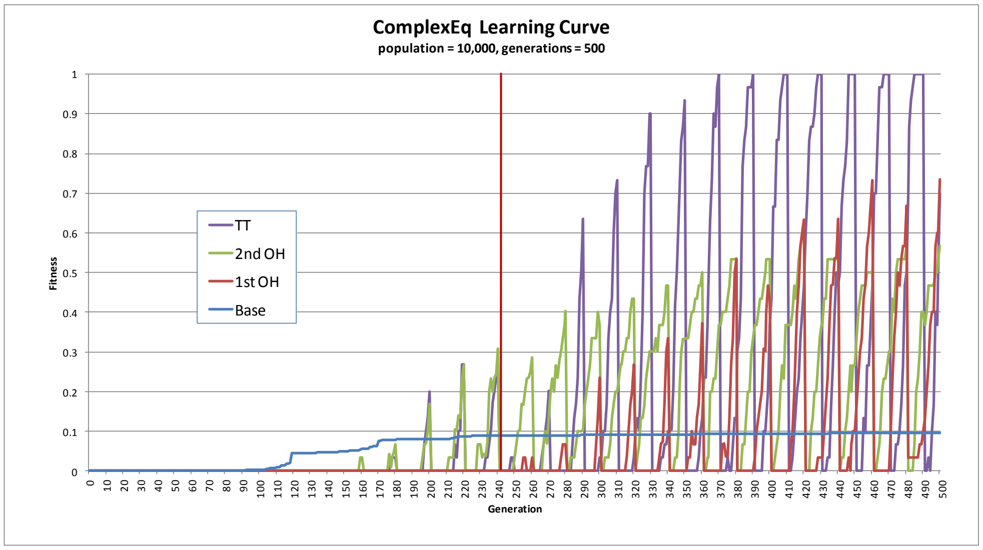

**Figure 36 - *ComplexEq* Average Fitness Curve using a Population of 10,000 versus Two-Tier Scheme using a Population of 10,000 seeding a Population of 5000**

The two-tier ACGP search methodology results in a far better average fitness than any of the other three methods using a population of 10,000 individuals. The advantages of the two-tier ACGP search method are apparent when the average execution times and average fitness scores are directly compared.

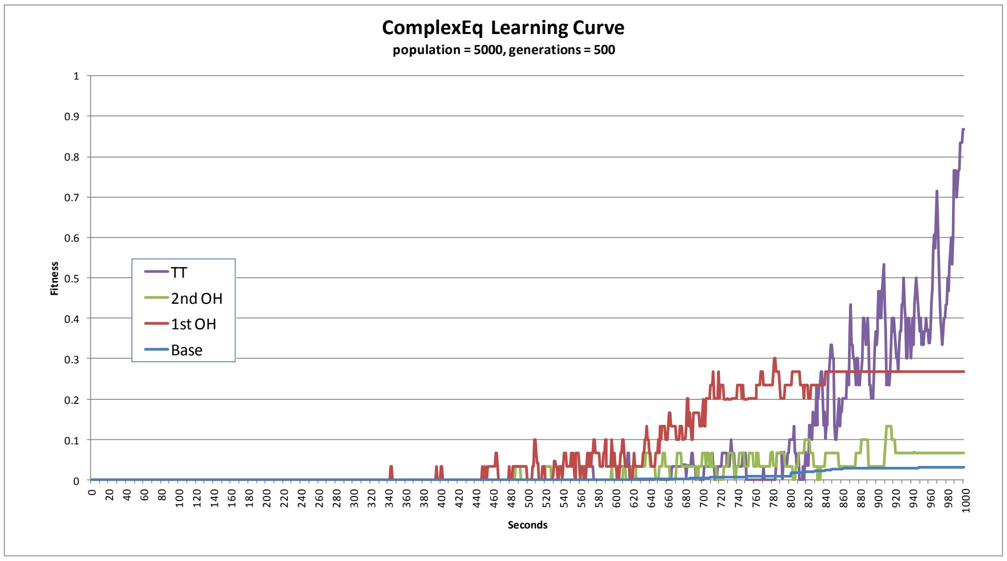

**Figure 37 - ComplexEq Average Fitness Curve using a Population of 5000 versus Two-Tier Scheme using a Population of 10,000 seeding a Population of 5000 compared on a time scale**

Figure 37 shows the learning curves from the experiments shown in Figure 35 substituting an x-axis representing time (1000 seconds) for one representing generations. The advantage of the two-tier scheme, in terms of fitness and execution time, is clear.

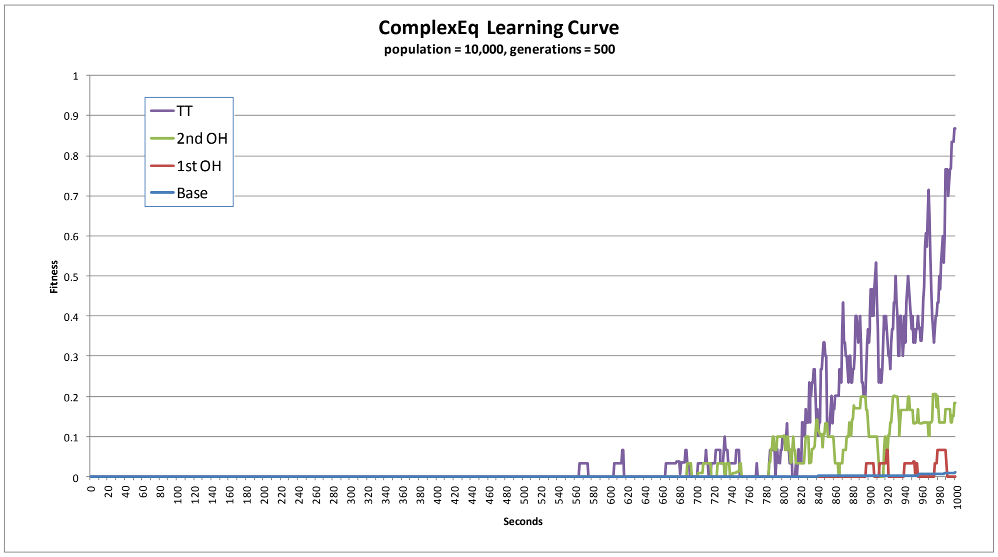

**Figure 38 - ComplexEq Average Fitness Curve using a Population of 10,000 versus Two-Tier Scheme using a Population of 10,000 seeding a Population of 5000compared on a time scale**

Figure 38 shows the learning curves from the experiments shown in Figure 36 substituting an x-axis representing time (1000 seconds) for one representing generations. The advantage of the two-tier scheme, in terms of fitness and execution time, is more distinct. Table 21 compares the average execution time for each set of runs for each search method. The two-tier search is fast and improves on most of the other searches. Only 1st order ACGP and 2nd order ACGP are faster than the two-tier search method. These observations are similar to those of the other GP search problems discussed earlier.

**Table 21 - Comparison of Average Execution Times (*ComplexEq*)**

| Execution Times | Population Size | | |
|---|---|---|---|
| | 5000 | 10,000 | 10,000 + 5000 |
| Base GP | 4657.00 | 8620.13 | - |
| ACGP 1st OH | 737.00 | 1559.27 | - |
| ACGP 2nd OH | 876.63 | 1676.33 | - |
| TT Aug (combined) | - | - | 1178.43 |

When the samples of execution time values for the set of independent runs of each method for this problem are compared with the sample of values for the two-tier method runs it is clear that these timing results are not the result of chance. The two-tier method is only slower than the two ACGP searches with the smaller population. Table 22 shows the p-values for the Mann–Whitney *U* test for each of these comparisons. The Mann–Whitney *U* test is used for these tests because it is a more robust statistic when attempting whether one distribution is stochastically greater than another.

**Table 22 - *ComplexEq* Execution Time Sample Significance (p-values, vs. the Two-tiered method)**

| Execution Times | | | |
|---|---|---|---|
| | Base GP | ACGP 1st OH | ACGP 2nd OH |
| 5000 | $3.334 \times 10^{-11}$ | $5.448 \times 10^{-11}$ | $3.472 \times 10^{-10}$ |
| 10,000 | $4.613 \times 10^{-10}$ | $4.969 \times 10^{-11}$ | $3.685 \times 10^{-11}$ |

Table 23 compares the average fitness scores for each set of runs for each search method on the *ComplexEq* problem. Here the two-tier search improves on the fitness scores from all of the other search methods. The strong fitness score of the two-tier search method offsets any execution time advantage 1st order ACGP or 2nd order ACGP had over the two-tier search in Table 21.

**Table 23 - Comparison of Average Fitness Scores (*ComplexEq*)**

| Fitness Scores | Population Size | | |
|---|---|---|---|
| | **5000** | **10,000** | **10,000 + 5000** |
| **Base GP** | 0.0330 | 0.0955 | - |
| **ACGP 1st OH** | 0.2669 | 0.7341 | - |
| **ACGP 2nd OH** | 0.1004 | 0.5675 | - |
| **TT Aug (combined)** | - | - | 1.000 |

When the samples of fitness score values for the set of independent runs for each method are compared with the sample of values for the two-tier method runs it is clear that these fitness results are considerable learning improvements and not the result of chance. Table 24 shows the p-values for the Mann–Whitney *U* test for each of these comparisons. The Mann–Whitney *U* test is used for these tests because it is a more robust statistic when attempting whether one distribution is stochastically greater than another.

**Table 24 - *ComplexEq* Fitness Score Sample Significance (p-values, vs. the Two-tiered method)**

| Fitness Scores | | | |
|---|---|---|---|
| | **Base GP** | **ACGP 1st OH** | **ACGP 2nd OH** |
| **5000** | $8.501 \times 10^{-13}$ | $1.004 \times 10^{-8}$ | $1.053 \times 10^{-11}$ |
| **10,000** | $3.678 \times 10^{-12}$ | $2.563 \times 10^{-3}$ | $5.619 \times 10^{-5}$ |

Although viable solutions for the *ComplexEq* problem require strict and complex functional structures that do not lend themselves to permutations that are also viable solutions, this two-tier search methodology can discover solutions with better fitness than standard GP techniques and do so more efficiently. An assumption that drives this performance is that the population for the first stage search by 1st order ACGP be large enough so structural diversity of the population ensures candidate solutions that can contribute useful building block information. Without this information, ACGP may bias the search toward suboptimal solutions.

# 6 Summarization and Conclusions

The empirical and theoretical results presented in this investigation demonstrate that genetic programming search of ACGP can be improved with a directed reduction of the representation space. This benefit mitigates issues associated with the normal over-specification of the components used to solve a given problem. The complete optimization of the set of functions and terminals is not necessary to achieve quality results for this methodology. The advantage in evolutionary learning can be gained with a coarse probabilistic reduction of the set of functions and terminals.

Chapter 1 described the factors that contribute to the normal complexity of a GP representation space. Past methodologies designed to optimize the search were discussed in Chapter 2. Adaptable Constrained Genetic Programming (ACGP) was introduced in Chapter 3. That discussion described how ACGP improves its evolutionary search by discovering desirable building blocks and probabilistically promoting their use. The limitations of ACGP learning was demonstrated using a regression example designed to illustrate these issues. Next, a case was made for a potential methodology to resolve these issues using a modification of normal ACGP operation. The general thesis of this research was stated in Chapter 4 with a description of a two-tiered modification of ACGP. This modified implementation of ACGP results in this operational process:

1) Run 1st order ACGP with typical parameter settings but with half the number of generations than normal (using multiple independent 1st order ACGP runs)
2) Review the final fitness scores for the set of ACGP runs
    a. If there are not a sufficient number of successful runs to combine into a seed heuristic model, increase the population size and repeat step 1
    b. Else, combine the final 1st order heuristic matrices for the successful runs by computing the arithmetic average of each individual heuristic to produce the seed heuristic matrix
3) Seed a set of 2nd order ACGP runs with the 1st order heuristic computed in step 2b
4) Analyze the final results from the two-tier search

This modification to normal ACGP processing capitalizes on several attributes of this application to improve its evolutionary search capabilities. ACGP can be run in one of three operational modes. Two of those modes (1st order heuristic mode and 2nd heuristic mode) provided the low-granularity and the high-granularity search processes used by the proposed scheme. ACGP supplies output of its discovered heuristics as probability tables. These probabilities can be presented to ACGP as input data that will seed and condition an ACGP search.

Finally, the empirical results presented in Chapter 5

a) Clearly describe the nature of the representation space search problem

b) Define the steps of a two-tier search methodology using ACGP

c) Establish the basis for the efficacy of this two-tier ACGP search scheme

d) Present the results of several experiments validating the efficacy and efficiency of this concept.

The results of the experiments in this investigation support the viability of this method in improving the ACGP search results over a complex representation space.

## 6.1 Summarization of experimental results

The empirical results described in this document clarify the issues associated with representation space complexity and the impacts they have on ACGP search. With a sample of weak solutions, ACGP will tabulate the most frequent building blocks and enhance their probability of selection. Unfortunately, without a sample of good solutions, this behavior becomes deceptive. ACGP will continue to tabulate the most frequent building blocks, irrespective of their contribution to solution fitness, and adjust their heuristic probabilities accordingly. These poor heuristics lead the ACGP search toward poor solutions.

The experiment with the *Bowl3neg* problem (Equation (13)) demonstrated how the strict-structure functions in this problem's target solution structure reduced the quality of a standard GP search and made this search more difficult than a corresponding search using only variable-structure functions in the target solution. Comparison of the desirable building blocks found in this problem with those found in the *Bowl3* problem (Equation (4)) explained the learning advantage of 2nd order ACGP has over 1st order ACGP for both problems but did not explain the fitness score differential between the two problems. The only potential explanation for this behavior by ACGP was the restricted structure of potential candidate solutions in *Bowl3neg*. Application of the two-tier ACGP search methodology on this problem confirmed that if the representation space was reduced better solutions can be discovered. Probabilistically biasing the selection of desirable building blocks and discouraging the selection of undesirable building block effectively reduces the representation space and thereby makes the search more efficient.

Additional experiments with the *artificial ant* problem and another more complex regression problem reinforced the assertions of the efficacy of the two-tier ACGP search methodology in improving ACGP search results. Both the *artificial ant* and the *ComplexEq* problem are characterized by candidate solutions with strict structure imposed by the desired functions used to form them. This structural constraint reduces the probability that a conventional GP search will be productive. The two-tiered ACGP search scheme will efficiently search the representation space with 1st order ACGP. This search develops a low-granularity probabilistic map of the representation space. This map probabilistically constrains the representation space and conditions the higher-granularity 2nd order ACGP search of the second tier to exploit more productive regions of the representation space. This constrained search scheme results in improved ACGP search results and efficient use of computation resources. In all the experiments, this search scheme achieves strong fitness results with increased efficiency in its execution time versus a base GP, 1st order ACGP, or 2nd order ACGP alone.

## 6.2 Implications of the Efficacy of this Methodology

The empirical results and analysis of the experiments in Chapter 5 demonstrate that a two-tiered ACGP search can be very effective. This two-tier search scheme combines the computational simplicity of a low-granularity method (1st order heuristics) with the quality resolution of a higher-granularity method (2nd order heuristics). This combined methodology exploits the strengths of the two techniques and mitigates their individual limitations. The low-granularity first stage search will develop a 1st order ACGP probabilistic model that constrains the original set of functions and terminals. This model will condition the higher-granularity search of the second stage using 2nd order ACGP. The conditioned second stage search will encourage the exploration of more productive regions of the representation space.

## 6.3 Limitations of the Specific Implementation of this Methodology

The experiments presented here show that employing ACGP in a two-tiered scheme improves its search capabilities for problems with complex structures. This two-tiered search methodology may not be a universal solution for all complex GP search problems. The *Boolean 11-multiplexer* was one problem that challenged this search scheme using ACGP. Analysis of the search results for this problem revealed that the scores for the typical fitness evaluation are contaminated by false positives. The output of a candidate solution is compared to the target output for the input data. It is considered a success if the two outputs match. No explicit verification is made to ensure that the correct input is presented as the output value. This contamination appears to also contaminate the ACGP heuristic analysis which relies on the assumption that there is dependency between the fitness of a candidate solution and its component building blocks. This result does not invalidate the principal assertions of the two-tier search scheme developed in this document. It does indicate the need for further work in refining ACGP's heuristic analysis and adjustment processing.

Unlike some of the other methods mentioned in Chapter 2 ACGP heuristics are not tied to specific locations in candidate solutions. This simplification helps the efficiency of ACGP processing but it can also complicate ACGP search when particular heuristics should be associated with specific locations in candidate solutions. Any refinement of heuristic locality information while retaining as much of the original processing efficiency could also benefit the proposed methodology.

Chapter 5.3 describes the steps of the proposed two-tier search scheme and its associated parameters. While the steps of this methodology are plainly stated the selection criterion for operating parameters such as population sizes for each tier, the number of training generations for each tier, or the preparation processing of the 1st tier's heuristics are not as clear and remain open questions.

## 6.4 Future development plans

While the results of this research support the original thesis that using ACGP in a two-tiered search scheme can reduce the complexity of the GP representation space and thereby improve ACGP's search process, much more work is necessary to explore the concept of probabilistic GP representation space reduction. Better component analysis can help improve the discovery of necessary and unnecessary functions and terminals in the specified component set. The ACGP heuristic analysis and adjustment process relies on the assumption that there is dependence between the fitness of a candidate solution and its component building blocks. When this assumption is invalidated, the heuristic processing becomes inefficient and potentially deceptive. This observation does not invalidate the principal assertions of the two-tiered ACGP search scheme developed in this document. It does indicate the need for further work in refining the heuristic analysis and adjustment process so that ACGP can conduct the two-tier search, or any search for that matter, in a more robust manner. These future research directions can build on the results of the work described in this document.

## Works Cited

1 Aleshunas, John. *Building Block Emergence in Genetic Programming (unpublished)*. University of Missouri - Saint Louis, Saint Louis, MO, 2011.

2 Angeline, Peter J., Pollack, Jordan B. Competitive Environments Evolve Better Solutions for Complex Tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93* (Urbana-Champaign, IL 1993), Morgan Kaufmann, 264 - 270.

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

4 Eiben. A. E., Smith, J. E. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin, 2003.

5 Janikow, Cezary Z. A methodology for processing problem constraints in genetic programming. *Computers & Mathematics with Applications*, Volume 32, Issue 8 (October 1996), Pages 97-113.

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

7 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.

8 Janikow, Cezary Z., Deshpande, Rahul A. Adaptation of representation in genetic programming. In *Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems, and Artificial Life (ANNIE'2003)* (Saint Louis, MO 2003), ASME Press, 45 - 50.

9 Janikow, Cezary Z., Mann, Christopher J. CGP Visits the Santa Fe Trail – Effects of Heuristics on GP. In *GECCO '05* (Washington, D.C. 2005), ACM.

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, Hauschild, Mark W. 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 Michalewicz, Zbigniew, Fogel, David B. *How to Solve It: Modern Heuristics*. Springer-Verlag, Berlin, 2004.

17 Ondas, Radovan, Pelikan, Martin, and Sastry, Kumara. Genetic Programming, Probabilistic Incremental Program Evolution, and Scalability. In *Advances in Intelligent and Soft Computing*. Springer, Berlin, 2006.

18 Pelikan, Martin, Goldberg, David E. Hierarchical Bayesian Optimization Algorithm. *Studies in Fuzziness and Soft Computing*, Volume 170/2005 (2005), 105-129.

19 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.

20 R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Creative Commons, San Francisco, 2008.

21 Salustowicz, Rafal, Schmidhuber, Jurgen. Probabilistic incremental program evolution. *Evolutionary Computation*, Vol. 5, No. 2 (1997), Pages 123-141.

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

23 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.

24 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).