

# STARS

University of Central Florida  
STARS

---


Electronic Theses and Dissertations, 2004-2019

---

2007

## A Neat Approach To Genetic Programming

Adelein Rodriguez  
*University of Central Florida*

 Part of the [Computer Engineering Commons](#)  
Find similar works at: <https://stars.library.ucf.edu/etd>  
University of Central Florida Libraries <http://library.ucf.edu>

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact [STARS@ucf.edu](mailto:STARS@ucf.edu).

---

### STARS Citation

Rodriguez, Adelein, "A Neat Approach To Genetic Programming" (2007). *Electronic Theses and Dissertations, 2004-2019*. 3323.  
<https://stars.library.ucf.edu/etd/3323>



# A NEAT APPROACH TO GENETIC PROGRAMMING

by

**ADELEIN RODRIGUEZ**

B.S. Computer Science, Florida International University, 2005

A thesis submitted in partial fulfillment of the requirements  
for the degree of Master of Science  
in the School of School of Electrical Engineering and Computer Science  
in the College of Engineering and Computer Science  
at the University of Central Florida  
Orlando, Florida

Fall Term  
2007

© 2007 ADELEIN RODRIGUEZ

## ABSTRACT

The evolution of explicitly represented topologies such as graphs involves devising methods for mutating, comparing and combining structures in meaningful ways and identifying and maintaining the necessary topological diversity. Research has been conducted in the area of the evolution of trees in genetic programming and of neural networks and some of these problems have been addressed independently by the different research communities. In the domain of neural networks, NEAT (Neuroevolution of Augmenting Topologies) has shown to be a successful method for evolving increasingly complex networks. This system's success is based on three interrelated elements: speciation, marking of historical information in topologies, and initializing search in a small structures search space. This provides the dynamics necessary for the exploration of diverse solution spaces at once and a way to discriminate between different structures. Although different representations have emerged in the area of genetic programming, the study of the tree representation has remained of interest in great part because of its mapping to programming languages and also because of the observed phenomenon of unnecessary code growth or bloat which hinders performance. The structural similarity between trees and neural networks poses an interesting question: Is it possible to apply the techniques from NEAT to the evolution of trees and if so, how

does it affect performance and the dynamics of code growth? In this work we address these questions and present analogous techniques to those in NEAT for genetic programming.

*To my parents Nancy and Jesus who have watched me grow and have always been there for  
me.*

## ACKNOWLEDGMENTS

I greatly appreciate the brainstorming sessions with my advisors Annie Wu and Kenneth Stanley who gave me the inspiration and insights that made this work possible. They guided me through the path of research and encouraged me to explore. I would also like to thank Avelino Gonzalez for his great advise which ultimately led me to discover an interest in this area of A.I.

## TABLE OF CONTENTS

LIST OF TABLES . . . . .	ix
LIST OF FIGURES . . . . .	x
CHAPTER 1 INTRODUCTION . . . . .	1
1.1 Motivation . . . . .	2
1.2 Contributions . . . . .	5
CHAPTER 2 FOUNDATIONS . . . . .	6
2.1 Genetic Programming . . . . .	6
2.1.1 Population Initialization . . . . .	7
2.1.2 Breeding (Genetic Operators) . . . . .	8
2.1.3 Benchmark Problem Domains . . . . .	8
2.1.4 Evolutionary Dynamics of Genetic Programming . . . . .	10
2.2 NEAT . . . . .	13
2.2.1 Encoding . . . . .	13



2.2.2	Historical Markings, Speciation, and Minimal Population Initialitation	14
2.3	Combining NEAT techniques and Genetic Programming . . . . .	16
2.3.1	Differences between NEAT and GP . . . . .	18
CHAPTER 3	APPROACH . . . . .	20
3.1	Encoding and Mutation . . . . .	20
3.2	Minimal Population Initialization, Speciation and Crossover . . . . .	22
3.3	Visualizing Change in Tree Structures . . . . .	27
CHAPTER 4	EXPERIMENTAL ANALYSIS OF NEAT-GP . . . . .	29
4.1	Experimental Setup . . . . .	30
4.1.1	Results . . . . .	31
CHAPTER 5	CONCLUSIONS . . . . .	49
LIST OF REFERENCES	. . . . .	51

## LIST OF TABLES

4.1	NEAT-GP parameters. . . . .	31
4.2	GP parameters. . . . .	32

## LIST OF FIGURES

2.1	<b>Standard encoding of an arithmetic program in GP using a tree representation.</b> . . . . .	7
2.2	<b>Standard swap crossover used in GP for tree representations.</b> . . . .	9
2.3	NEAT genotype representation example [SM02] . . . . .	14
3.1	<b>Tree and list encoding of two trees created consecutively.</b> Innovation ids are assigned to nodes and connections between nodes. . . . .	22
3.2	<b>Grow mutation.</b> . . . . .	23
3.3	<b>One node mutation.</b> Three consecutive mutations in which two nodes are changed. Note that genome 1 and genome 3 have a connection with innovation number 2 although their source and target nodes differ. However, the same connection ids are kept to note to positional homology. . . . .	24
3.4	<b>Two possible crossover scenarios.</b> In the first crossover matching connection 2 is inherited from parent 1 thus connections 5 and 6 are deleted from parent 2's list since they hang from the rejected connection 2 of parent 2 and also are not common between parent 1 and 2. . . . .	27

3.5	<b>Two example tree visualizations.</b> Bluer colors reflect older node and branches. . . . .	28
4.1	Data sets 1 and 2 from Dow Jones Industrial High Values. . . . .	36
4.2	<b>Example snapshot of species in two generations.</b> Species with similar fitness but which differ in structures and the total number of nodes coexist in the population. . . . .	37
4.3	<b>Example of two trees with similar fitness but different structures.</b> The tree on the right belongs to a specie that beat the specie on the left. . .	37
4.4	<b>Run summary of experiment 1 using NEAT-GP to approximate data set No. 1.</b> Highest fitness corresponds to a value of 1. No exact solution was found thus the bottom right diagram shows generations where solutions were found as -1. . . . .	38
4.5	<b>Run summary of experiment 1 using regular GP to approximate data set No. 1.</b> Highest fitness corresponds to a value of 1. No exact solution was found thus the bottom right diagram shows generations where solutions were found as -1. . . . .	39
4.6	<b>Summary of run 2 where the best structure was found in experiment 1 using NEAT-GP to approximate data set No. 1.</b> This individual has a fitness of 0.684 and was found in generation 1998. . . . .	40

4.7	<b>Summary of run 19 where the best structure was found in experiment 1 using GP to approximate data set No. 1.</b> This individual has a fitness of 0.58. . . . .	41
4.8	Experiment 1: Best approximation found of data set No. 1 using standard NEAT-GP (left) and standard GP (right). . . . .	42
4.9	<b>Run summary of experiment 1 using NEAT-GP to approximate data set No. 2.</b> Highest fitness corresponds to a value of 1. No exact solution was found thus the bottom right diagram shows generations where solutions were found as -1. . . . .	43
4.10	<b>Run summary of experiment 1 using regular GP to approximate data set No. 2.</b> Highest fitness corresponds to a value of 1. No exact solution was found thus the bottom right diagram shows generations where solutions were found as -1. . . . .	44
4.11	Experiment 1: Best approximation found of data set No. 2 using standard NEAT-GP (left) and standard GP (right). Fitness of the individuals encoding these solutions were 0.80 and 0.93 respectively. . . . .	45
4.12	<b>Example run (run 5) from experiment 1 using data set No. 2 and GP.</b> This run shows a high average number of nodes of the population as well of the best individual which does not match its corresponding slow fitness increase. . . . .	46

4.13	<b>Run summary of experiment 1 using NEAT-GP for the symbolic regression problem of the quintic equation.</b> Highest fitness corresponds to a value of 1. All but 8 runs found a solution. . . . .	47
4.14	<b>Run summary of experiment 1 using GP for the symbolic regression problem of the quintic equation.</b> Highest fitness corresponds to a value of 1. All but 17 runs found a solution. . . . .	48

# CHAPTER 1

## INTRODUCTION

The purpose of this work is to apply some of the techniques from the NEAT algorithm to Genetic Programming and answer the following questions: *1. What are the effects on bloat?* *2. What are the effects on fitness?* Several measures of bloat such as the amount of invalidator or redundant code, number of nodes, and depth have been defined in order to investigate two major problems in GPs. The first problem is that the average size of the individuals in the population grows excessively in size and depletes computing resources without an equivalent increase in fitness [LP06]. Secondly, the solutions found tend to be very large. To address our first question, we focus on bloat as measured by the **number of nodes and depth of trees** and look at these values in the average population as well as in the best solutions found. Since changes to the dynamics of bloat in GP are likely to affect fitness, the second question follows.

## 1.1 Motivation

NEAT uses the concepts of historical markings and speciation that showed effective for identifying similarities between genotypes and clustering them so that mating occurs between individuals with homologous genetic information. Some of the components present in NEAT have been somewhat explored separately in GPs although to my knowledge no work has investigated how these components can work together. Several techniques have been created to understand the dynamics of GPs as well as to incorporate this knowledge into better genetic operators which can for example increase diversity in the population. This work has been motivated by the general notion that increasing diversity in the population usually leads to improvements in fitness for evolutionary computation algorithms. Some of these techniques mark nodes using unique identifiers to keep track of ancestry such that by inspecting these ids (and thus the trees ancestry) it is possible to compare their structures.

Other work has looked at using species or demes based on genotypical as well as phenotypical similarities of trees. In GPs the genotype corresponds to the content and structure representing the individual while the phenotype refers to the behavior (fitness) of the individual when evaluated. The purpose of speciation in the GP literature has been to encourage mating between similar or dissimilar individuals, specifically mating individuals with similar genotypes or individuals with different phenotypes. Crossovers based on genotype similarities (homologous crossovers) have also been tested which have taken into account combinations of tree size, node content and location when recombining parents information. Although



different flavors of speciation, ancestry markings and homologous crossover have each shown to provide some performance improvement in GPs, we do not know the effects of combining them into one algorithm. NEAT presented a principled explanation to why the dynamic between these components is beneficial:

1. Speciation allows for similar structures to be rewarded or penalized together. They provide a niche for structures to develop their potential without being subject to drastic penalization and also for mating with similar structures.
2. Historical markings provide a way to compare structures and thus allow speciation and crossover based on structure.
3. Initializing the population with small structures provides a way to differentiate structures at an early stage with historical markings as well as avoids searching for solutions in unnecessary large spaces.

One of the possible benefits we see is that given that speciation groups trees by similarity, some of the species are likely to contain individuals which are bloated. In essence, a specie could contain individuals with similar types of bloat. We believe that this type of speciation based on structure could be used to isolate different types of bloat and controls their spreading or remove them from the population. NEAT penalizes and rewards individuals using fitness sharing so that the fitness of an individual is reduced depending on how many other individuals fall in the same specie. This leads to a single specie not being able to take over

the population which can indirectly keep the different types of bloat under control. Tied to the use of speciation is that of historical markings which could in itself be useful in GP for comparing both position and content information of trees. Structural parts that have the same marking are guaranteed to be in the same place in the tree and possibly have the same information, although it is possible to have equal structures with different markings.

Also, taking a NEAT approach at genetic programming could possibly change the dynamics of evolution so that not so many runs are needed. In the GP literature it is most common to find experiments of short runs instead of long ones for many of its benchmark problems [Luk01]. This has been justified by several factors: convergence of the population for some problems in a few number of generations and also tradeoffs between continuing a run with a bloated population (which exhausts computing resources) instead of starting a new run. GPs rely greatly on the genetic material present in the initial population and differently than other evolutionary algorithms, its evolutionary mechanism does not lend to incrementally finding or building solutions for larger number of generations (i.e. bloat hinders long evolutions). NEAT on the other hand, does not require many runs to find a desired structure since it has a systematic approach for gradually building solutions from simpler structures first. If NEAT concepts are beneficial when applied to GPs one can expect longer runs will be necessary to find solutions but with a potential to solve more complex problems.

Given the difference in representation and domain of GP and the problems NEAT was applied to, we expect to face some issues while applying these concepts to GP. One difference is that small changes in structure affect fitness greatly in some problems such as symbolic

regression which is the problem studied in this work. This could cause uncertainties while implementing speciation for GPs which depends both in fitness and in structural similarity. Although the fitness landscape is continuous in this problem, non functional code and precision wrappers around operators are some of the factors that account for very different trees having the same fitness [BGK04]. If drastic changes in fitness caused by small tree alterations happen often, then an explosion of species dying and being created is expected. Another issue we foresee is that NEAT uses a seed individual to create the entire initial population. This is based on the assumption that there is a seed topology which can always be mutated into a solution and also based on the fact that inputs and outputs are not mutated so that in the specific case of ANNs only a hidden layer is allowed to evolve. However, in the domain of GP trees there is no single tree structure from which all solutions can evolve, the only known fact about the structure is that a tree has a single output.

## 1.2 Contributions

The most important contribution of this research is a study of the impact of applying principled concepts from the evolution of neural networks using NEAT to the evolution of trees in genetic programming. We have created an evolutionary dynamic similar to NEAT and shown how code growth (number of nodes) is inherently controlled by the tendency of the algorithm towards minimalism. This work sets the ground for further work to better exploit the benefits of the NEAT approach in genetic programming.

## CHAPTER 2

### FOUNDATIONS

#### 2.1 Genetic Programming

Genetic programming (GP) is a mechanism for evolving computer programs for approximating a solution to a particular problem or for exploring possible programs in a domain. The algorithm has an initial population of randomly generated programs built of functions (eg. arithmetic, boolean, domain specific) and terminals (eg. domain specific, numerical). In GP programs have been commonly represented as trees ( figure 2.1) although other representations such as linear and stack datastructures have also been used. As an evolutionary algorithm, GP uses genetic operators such as mutation and crossover to combine and modify the information of the individual programs. The lifecycle of the algorithm is as follows:

1. Initialize population of programs
2. Evaluate fitness of programs
3. While the stopping criterion is not met
  - (a) Select individuals to breed new population

- (b) Create new individuals using breeding operators
- (c) Evaluate population fitness
- (d) Select individuals to breed new population

4. Return highest fit individual

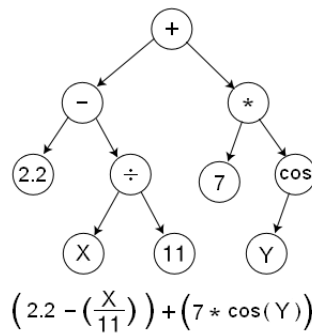


Figure 2.1: **Standard encoding of an arithmetic program in GP using a tree representation.**

### 2.1.1 Population Initialization

The most common initialization algorithms used by the GP community as described in [Koz92] are Full, Grow, and Ramped Half-and-Half. Given a maximum depth  $D$ , Full grows a tree by picking a random function for nodes if the current depth does not exceed  $D$ , or else a terminal is selected. The Grow method differs in that at any depth within  $D$ , a terminal as well as a function can be chosen. The Full algorithm creates trees of the same size since it grows trees up to the maximum size provided while the Grow algorithm creates trees of various sizes.

Ramped Half-and-Half selects a random depth value within the specified depth range and within an equal probability it uses the Grow algorithm, otherwise it uses the Full algorithm to create a tree [Koz92]. This last method is the preferred one used in the GP literature since it produces trees with a wide variety of sizes.

### **2.1.2 Breeding (Genetic Operators)**

The standard GP crossover operator combines information from the parent programs and produces two offsprings. Parents are combined by selecting a point within each parent's tree uniformly and swapping the subtrees under the crossover point as seen in in figure 2.1.2. Selected subtrees could be as small as containing a single terminal. A special case to note is when the crossover point in both parents is at the root, then the offsprings are merely a copy of the parents. Also, when an individual is mated with itself, the children are not necessarily copies of the parent since crossover points may differ for the two parents. Standard mutation consists of replacing a subtree with a newly created one.

### **2.1.3 Benchmark Problem Domains**

Some of the benchmark problems mentioned in the literature are symbolic regression, artificial ant, multiplexer, lawnmower, and even or odd parity. Symbolic regression seeks to find

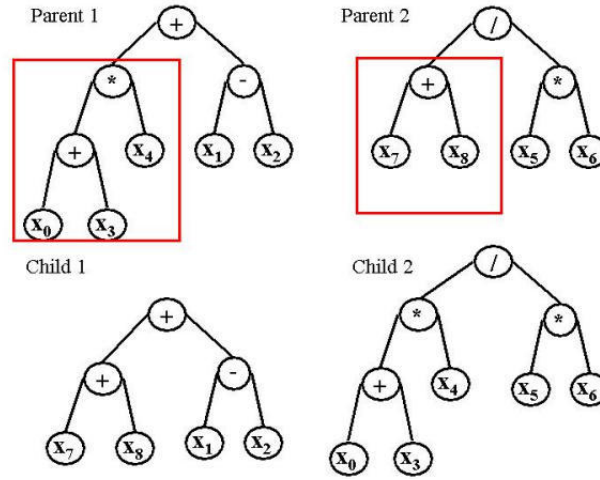


Figure 2.2: **Standard swap crossover used in GP for tree representations.**

a mathematical expression that corresponds to a set of domain and range data points. Its purpose is to find the symbols as well as coefficients that form the equation as opposed to other regression mechanisms such as linear regression that only involves finding the coefficients [Koz92]. The common regression equations used are  $x^4 + x^3 + x^2 + x$ ,  $x^5 - 2x^3 + x$ , and  $x^6 - 2x^4 + x^2$ . A GP attempts to find a tree which best approximates the data points produced by these equations given a function set such as  $F = +, -, *, \%, \text{SIN}, \text{COS}, \text{EXP}, \text{RLOG}$  and a terminal set  $T = x$ . The data points against which the fitness of the evolved tree will be evaluated consist of 20 domain and range values generated from the quartic equation within the real interval of  $[-1.0, 1.0]$  [Koz92]. A raw fitness value (equivalent to the standard fitness in this case) is computed for each tree by evaluating it over the 20 data points and summing the difference of its output and the output from the data points.

In the ant trail problem the goal is for an ant to travel along an irregular path and find all available pieces of food. The ant is evaluated according to the amount of food it finds as

well as the time taken. The ant is allowed to perform the following actions: turn right or left, move, or do nothing (no-operation). The Santa Fe trail is one of the commonly used trails for the ants to travel and is an irregular trail with several types of gaps and with 89 food items which pose a complex problem.

The parity problem takes as input a random number of strings of 0's and 1's and outputs whether the strings have an even number of 1's, or in the case of the odd parity problem whether there is an odd number of 1's. If the problem is even 4 parity, then the goal is to find a function that best guesses  $2^4$  combinations of 4-bit length strings. The function set usually consists of the binary operators *and*, *or*, *nand* and the terminal set is formed by a number of inputs corresponding to the length of the strings.

#### **2.1.4 Evolutionary Dynamics of Genetic Programming**

It is important to note which are the characteristics that have been found to hinder the performance of genetic programming in general. One of these is its convergence towards less structural diversity for which it has been compared to a hill-climber algorithm [MH99], [PL98]. The evolution is characterized by an initial period of exploration in the first generations and a second phase of exploitation of a structure. In [BGK04] it was noted that while all runs perform this exploitation, they do not always exploit structures which can be likely improved. Thus some methods that extend the exploration period such as dynamic mutation rates, demes (species) and island models have been used to extend the exploration



period. Another property of GPs and the tree representation is that some structures are easier to evolve than others in GP [DLT03].

Also, in genetic programming is common to observe weak causality, where small changes in the genotype of individuals in the population cause drastic changes in fitness. This is a result of the representation and causes instability in the search [RB95]. Some problems such as regression problems have shown little correlation between structural and fitness diversity [BGK04] so that it becomes difficult to understand when in evolution is diversity more or less beneficial.

Unjustifiable and uncontrolled code growth has also been widely documented as a problem and is an element we will monitor in our results. Currently, there exist three theories of code bloat. A first theory holds that code bloat is caused by introns, which are subtrees that can be removed from the program without affecting the result of the program and thus do not affect fitness. Introns have been separated into two categories. The first type of intron describes those subtrees that contain unoptimized code, that is code that can be simplified without affecting the program's outcome (e.g.  $(* x 1)$ ,  $(- x x)$ ). A second type of introns, or inviable code, describes subtrees that cannot possibly be replaced by any code that would affect the fitness of the program since there exists another subtree that is already invalidating this intron's effect (E.g.  $(* x 0)$ ,  $(-x x)$ ). There are three main existing theories of how introns spread in the population and thus contribute to bloat. The hitchhiking theory [Tac94] states that if introns are part of relatively fit trees, then it is probable that crossover gives a free ride to introns. Another theory holds that introns are propagated because they

protect an individual against the destructive effect of crossover by providing more neutral crossover points [BT94], [NB95], [MH99]. A similar theory, of removal bias, suggests that the mechanics of the algorithm have a bias towards removing small trees which do not exceed the size of the inviable subtrees but has no bias for the replacing tree, thus more often adding subtrees that are larger than those removed [SFD96]. More recent theories state that bloat is caused by other factors other than introns, such as fitness. The theory which states that fitness causes bloat is based on that there is a greater probability to find a larger tree than a semantically equivalent one which is smaller. Thus, when a fit but large solution is found it is kept since there is no pressure to find another smaller solution [LP98a].

Several approaches have been attempted in order to reduce bloat. A common and early method sets a limit on the size of the trees and only individuals that fall under this size are accepted into the population [Koz92]. Under this method, the crossover and mutation operations are retried a number of times until it generates a tree of valid depth. Another approach is parsimony pressure which takes into account the size of the trees in the selection step. Parsimonious pressure has been applied by combining fitness and size into one value (parametric method) and by considering them as separate values (nonparametric). In the parametric parsimony method the size is usually combined with another factor for fitness evaluation. However, a problem that arises is that through evolution the importance of size changes but it is difficult to capture it in an equation combining fitness and size. Nonparametric parsimony has been applied in different ways, for example by first comparing the

fitness and if these match then sizes are compared [LP02]. For example, if two individuals have the same fitness, the one with less number of nodes or smaller depth is selected.

## 2.2 NEAT

Neuroevolution of Augmenting Topologies (NEAT) is a method for evolving the topologies of Artificial Neural Networks (ANNs) of increasing complexity [SM02]. In the following sections we describe the encoding used by NEAT and the three main components that make this approach successful.

### 2.2.1 Encoding

In NEAT networks are encoded using a list of *connection genes* which represent how *node genes* are connected to each other as depicted in figure 2.2.1. These two different types of genes are the building blocks which are manipulated through evolution. Connection genes describe the weight of the connection, the incoming and outgoing nodes, an innovation number, and whether the connection is enabled. These genes could be disabled through mutation but they remain in the genome.

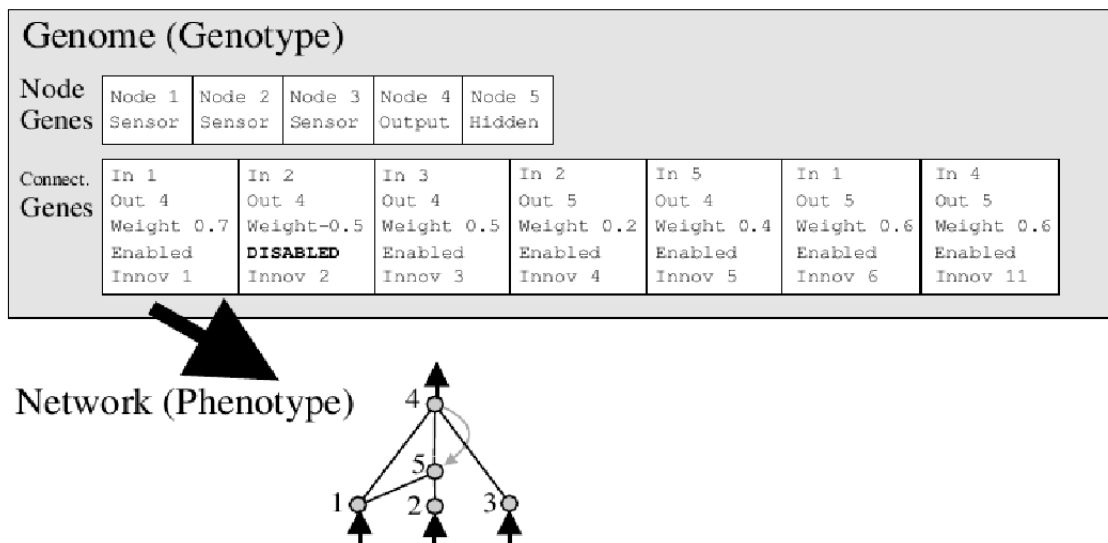


Figure 2.3: NEAT genotype representation example [SM02]

## 2.2.2 Historical Markings, Speciation, and Minimal Population Initialization

NEAT uses the history of evolution of genes to determine compatibility between genomes. Node genes, as well as connection genes, are given an innovation number which marks when in evolution the gene structure appeared. This number is assigned in chronological order as genes are created. Within a generation, a list of connection genes is kept and if consecutive mutations create a connection containing the same source and destination nodes, the same innovation number is assigned. This eliminates cases where the same structural discoveries are assigned different innovation numbers.

Historical markings not only allow for insight into the population diversity, but they also allow for comparison of structures and for performing homologous crossover. Crossover works

by separating matching genes and nonmatching ones. Nonmatching genes are separated into those genes that are present in one parents but are missing in the other parent (disjoint genes), and those that differ only at the end of either parent's connection genes (excess genes). The offspring is built by aligning the matching genes and randomly choosing from either parents, then adding the excess and disjoint genes from the fittest parent.

The population is initialized by creating a seed individual of minimal structure containing only inputs and outputs, then mutating this individual to create the rest of the population. Genes are given historical markings as they are created. With this approach, evolution starts with genomes of a common ancestor and thus share some historical information. Also, by starting with small networks allows for structures to be built incrementally thus moving from low to higher dimensions.

In order to allow smaller genomes time to be optimized, NEAT uses species which provide a niche for initially not so fit genomes to improve and compete with similar genomes. Species group genomes with similar topologies. Initially, a random representative genome is chosen and the first set of bred offspring is compared against it. If the differences are within a certain compatibility threshold then the offspring is placed into the same species as the representative. Otherwise, a new species is created for the offspring. In the following generations, a random genome inside each species is chosen as its representative. At reproduction time, every species is assigned a number of offspring proportional to the fitness and number of members it has. NEAT was initially tested on several benchmark problems of different difficulty levels such as the XOR and double pole balancing problems and consistently found solutions for these

problems more efficiently than existing methods [SM02]. Each of the three main components of the algorithm explained above were independently shown to be necessary to make NEAT work. When testing the contribution of speciation an important observation noted was that in the absence of speciation and minimal topology initialization, the population has a tendency to quickly converge to an initially fit individual thus hindering both performance and the ability to find a solution. Also, without speciation the best performing networks found in the beginning were larger than needed because potential networks of smaller size were not allowed to stay in the population enough time.

## **2.3 Combining NEAT techniques and Genetic Programming**

### **2.3.0.1 Historical Markings**

Previous work explored the used of markings to study the syntactic diversity in genetic programming and how crossover affects it [MH99]. In this approach, tree nodes were assigned an ID and memID when created, and these numbers remained identical until crossover. The memID changed through evolution to reflect when the subtree under a node had changed. When one point standard swap crossover was performed, in each of the child trees all the nodes in the path to the changed subtree received a new memID to indicate the change although their IDs remained intact. This way it was possible to know which nodes were copies of the same node by comparing their memIDs and which ones had the same ancestry by

comparing their IDs. Through this methods they showed how the final population descended from a few individuals and in one extreme, from only one individual. From their experiments it was noted that using standard genetic programming parameters out of the initial nodes only less than 1 % was present in the last population. Also, only seven from 500 individuals contributed to the genetic material in the final individuals.

### **2.3.0.2 Homology**

Studies on standard GP crossover, which swaps subtrees from two differen parents, have shown that it has a negative effect on the offsprings. One of the main documented reasons is that it does not preserve the context of promising code segments. Factors such as the location in the tree and the arity of the subtree being swapped are ignored by this crossover. Intuitively, the worth of subtrees that have survived may depend on some of these factors that constitute its context.

Among the most promising alternatives to standard crossover is homologous crossover which attempts to preserve the context of code segments. This approach has a biological basis on how the exchange of genetic information happens between two parents. In nature, there is a bias towards exchanging genes in the chromosome that are in the same position and encode the same type of properties, for example when deciding the hair color of a child only genes encoding hair color information in both parents parents are exchanged. Speciation is one phenomena that allows homologous crossover by restricting mating between individuals of

chromosomes that are largely similar. Also, during DNA recombination, strands of DNA are aligned so that crossover recombines regions that are similar or identical [WHR87]. [FCB99] noted that standard genetic programming has a harmful type of emergent homology which is caused by code bloat. When code is bloated, crossover is likely to swap introns which are similar in that they not contribute to fitness or hinder other subtrees from contributing to fitness. This leads to the formation of a species of bloated trees with position independent homology. Previous works on designing homologous crossover mechanisms include [Dh94] who restricted crossover to only between nodes that are in the same position in both parent trees. Also, [PL97] experimented with a one point crossover between nodes of same position and arity in the trees. This method showed improvements early in evolution but not later on. [FCB99] created a “sticky” crossover for a linear GP representation that selected a code region in one program and exchanged it with a region of the same position and length in the other parent thus increasing the likelihood of exchanging functionally and semantically similar code. When compared with standard GP crossover “sticky” crossover showed some improvement over a range of several problem domains [Han03].

### **2.3.1 Differences between NEAT and GP**

The tree structures imposes a dependency between connections so when one connection is removed, the subtree that it connects to must be removed as well. Thus in the process of



connection gene alignment during crossover, once a decision is made to not add a specific connection (E.g. a disjoint connection from a less fit parent), another number of connections which depend on the connection in question may have to be removed from the list as well. In NEAT there is not such issue, connections can be removed without dependencies needed to be taken care of.

Also, NEAT uses a seed individual to create the entire initial population. This is based on the assumption that the seed individual can always be mutated into a solution and also based on the fact that inputs and outputs are not mutated so that only a hidden layer is allowed to evolve. However, in NEAT-GP there is no single tree structure from which all solutions can evolve, the only set known fact about the structure is that a tree has a single output.

## CHAPTER 3

### APPROACH

In this chapter we describe how the main mechanisms in the NEAT algorithm (historical markings, speciation, and minimal population initialization) were applied to a GP algorithm. Our implementation uses the Evolutionary Computation in Java (ECJ) system and adds the appropriate extensions.

#### 3.1 Encoding and Mutation

In GP-NEAT trees were represented using a traditional recursive tree structure as well as a linear structure. Trees are created using a ramped Half-n-Half algorithm and innovation numbers or ids were assigned to nodes in the trees in consecutive order as they are attached to the tree. A simultaneous list is built which holds connection genes that represent the linkage between nodes as seen in figure 3.1. Connection genes contain a source and a target node and are also assigned innovation numbers. These two structures facilitate different operations on the genome such as crossover as will be explained later on. When a node and its two corresponding connections are created, these connections remain the node's connections to

its children as long as the node exists. Note that trees of equal semantic content can have different nodes and connection numbering. However, if two nodes have the same innovation numbers they appear in the same position in the tree always thus no two nodes with same innovation numbers can appear in different parts of the tree. Similarly to NEAT, this allows a way to compare structures by comparing their innovation ids and although two connections with equal innovation numbers might link different source and target nodes, their similarity in ids reflects a position similarity in the trees. Trees with similar connection and node genes evolved from a common ancestor.

We have devised three types of mutations: grow mutation, one node mutation, and delete one node mutation. Grow mutation is the traditional GP mechanism of selecting a random node in the tree genome and replacing it with newly grown subtree. We add an extra step for marking nodes and connections of the new subtree as depicted in figure 3.1. When the new subtree is connected to the tree, the innovation number of the connection leading to this new subtree root remains unchanged. We do not make changes to nodes if only terminals are changed based on the assumption that a change in terminal is usually not a drastic change which requires a note in change of structure. One node mutation simply replaces the operator or terminal of a node as was used by [Che98]. The new node is assigned the same innovation number as the node that is being replaced. One node mutation is shown in figure 3.1. Finally, nodes can be deleted and replaced by a terminal. Note that the first type of mutation, grow mutation, can also serve as a way to delete structures in cases when the new subtree is smaller than the replaces one. Through these mutations it is possible

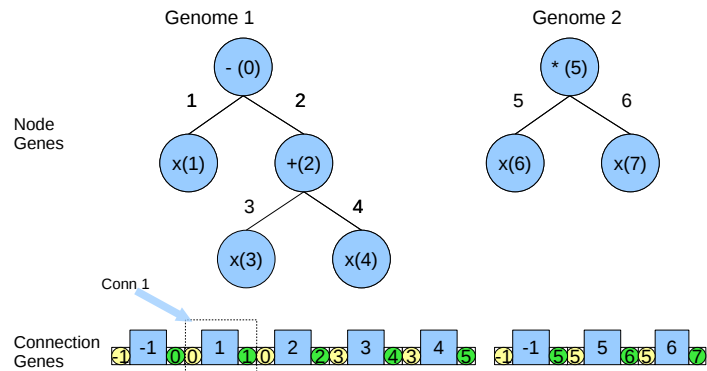


Figure 3.1: **Tree and list encoding of two trees created consecutively.** Innovation ids are assigned to nodes and connections between nodes.

that connections with the same innovation numbers loose their homology in both source and target nodes, the connection still represents a marker for positional homology so that later during recombination similar parts of the tree are exchanged when connections are aligned. Mutations are applied with the same probabilities to terminals, nonterminals and the root.

### 3.2 Minimal Population Initialization, Speciation and Crossover

Differently from NEAT, our approach does not use a seed genome to create the initial population although we do create a population of random small genomes. In the NEAT algorithm the initial population is generated by creating a seed network of minimal structure and then mutating it to create the rest of the population which served two main purposes. First, this was intended to avoid searching in unnecessary large spaces by starting to explore the space

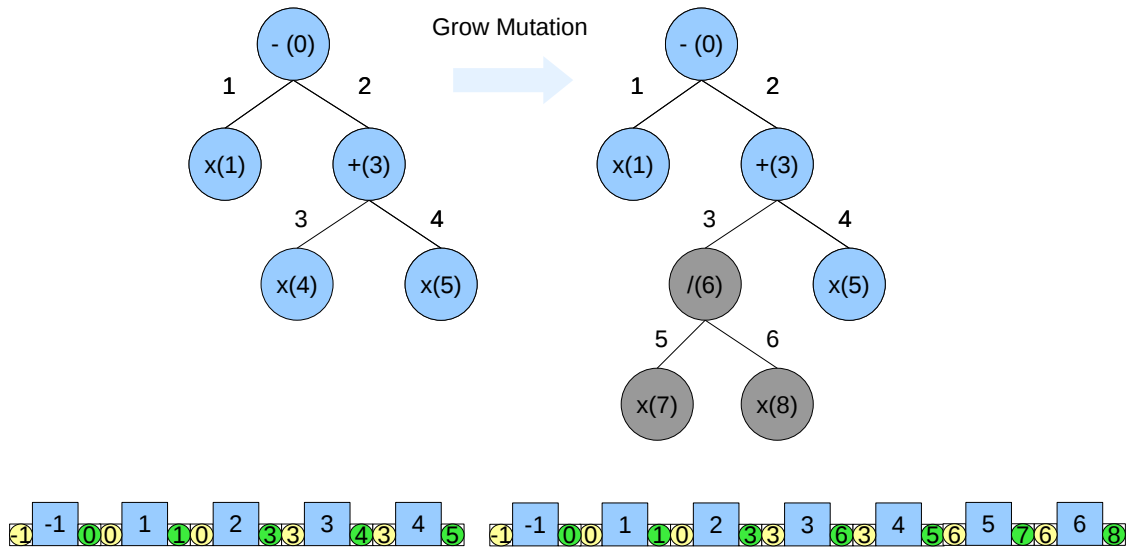


Figure 3.2: **Grow mutation.**

of small structures and gradually move on to higher dimensional spaces. Second, given that if the same mutation happens more than once in a generation then it is assigned the same historical marking, relative similarities and differences between building blocks in genomes are recognized early in evolution. The seeding mechanism relies on that some network structures which can be used as a seed and from which it is possible to evolve almost all if not all structures desired. However, in the domain of GPs, it is not clear if there exists such structure which can be used as a seed to the initial population. Some of our attempts at using a seed showed that some tree structures can be greatly harmful more than others therefore further research should be done if a seed is to be used. In our work we attempt to fulfil the first purpose of avoiding search in unnecessary spaces by starting with small structures but do not use a seed. Instead, we create initial random trees of maximum and minimum depth

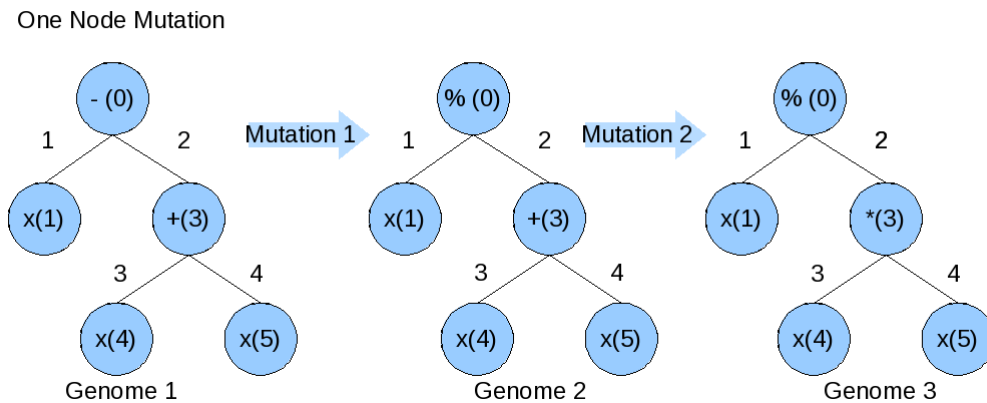


Figure 3.3: **One node mutation.** Three consecutive mutations in which two nodes are changed. Note that genome 1 and genome 3 have a connection with innovation number 2 although their source and target nodes differ. However, the same connection ids are kept to note to positional homology.

2 and 1 correspondingly.

Crossover is directly performed on the list of connections and produces one offspring from two parents. Because connections are maintained in sorted order by their innovation ids, both parents lists are easily aligned for recombination. Differently than in NEAT, here we combine the concepts of excess and disjoint genes into one category so that we only have two types of connection genes: matching and nonmatching. Matching connection genes are those that merely have the same innovation number and are inherited randomly by the offspring. These allows homologous recombination where tree content in the same positions and with possible similar nodes are exchanged between parents. Nonmatching genes are inherited from the most fit parent. In both cases, when one parent's connection is chosen over another one the rejected connection must be deleted from the list and all the corresponding connections "hanging" from it which are not present in the other parent must be deleted as well since

they are irrelevant for further matching. If crossover is performed between two genomes that do not have any connections in common, then one of the parents is chosen randomly to be the resulting offspring which does not provide any beneficial recombination. This is not a desired case and is minimized by speciation.

Speciation is one of the key components of the NEAT algorithm for it is intended to protect new genomes which can potentially contribute important innovations by allowing them to stay long enough in the population even if they are not highly fit. Also, speciation allows for crossover to take place between similar individuals and thus allowing it to take advantage of homologous recombination. In our algorithm this is accomplished as in NEAT by placing individuals in species with similar individuals and evaluating an individual's fitness based on how big is the niche it shares with others. Each generation, the fitness of every species is computed and a lowest fit percent 20% is removed from each species. The remaining individuals will mate and mutate within their species to create the next population. The elite individuals from every species are copied to the new population intact if the species contains more than 2 individuals. This restriction is applied so that species do not remain alive by the sole existence of its elite individual.

Each species is assigned a number of offspring's  $s_{offspring}$  it can generate based on its fitness average fitness  $\bar{F}_s$  and the number of individuals in the species as shown in equation (3.2). This equation is used in NEAT and derived from the explicit fitness sharing equation presented by [GR87]. Once a new population is bred, individuals are placed into their corresponding species by computing the individual's compatibility with representatives from

each specie in the previous generation. A single representative is chosen randomly from each specie. The number of offsprings of a specie does not necessarily represent the number of individuals a specie will have in the next generation. When the offspring is too structurally different from the representative of the specie, it will be placed in a different specie instead.

$$s_{offsprings} = \frac{\bar{F}_s}{\bar{F}_{tot}} |Population| \quad (3.1)$$

$$\bar{F}_{tot} = \sum_j^n \bar{F}_j : \text{Total of species averages.} \quad (3.2)$$

Compatibility between genomes is computed by adding the number of nodes which differ between genomes and comparing it is against a *compatibility threshold*. In NEAT compatibility is based on the connections genes rather than node ids. In part this is because connection genes in NEAT are structures which can be mutated as well (their weights can be mutated) so comparison between connections is also necessary. However, because our connections gene only encode source and target nodes and our node encoding guarantees that two nodes with the same id appear in the same position in the tree and can potentially have the same functionality, comparing nodes is an accurate strategy for comparing trees. The compatibility threshold is incremented or decremented dynamically every generation in order to maintain a certain number of species. To counteract specie stagnation NEAT penalized species that have not improved their average fitness for a certain number of generations by decreasing their fitness so that its number of assigned offsprings is reduced.



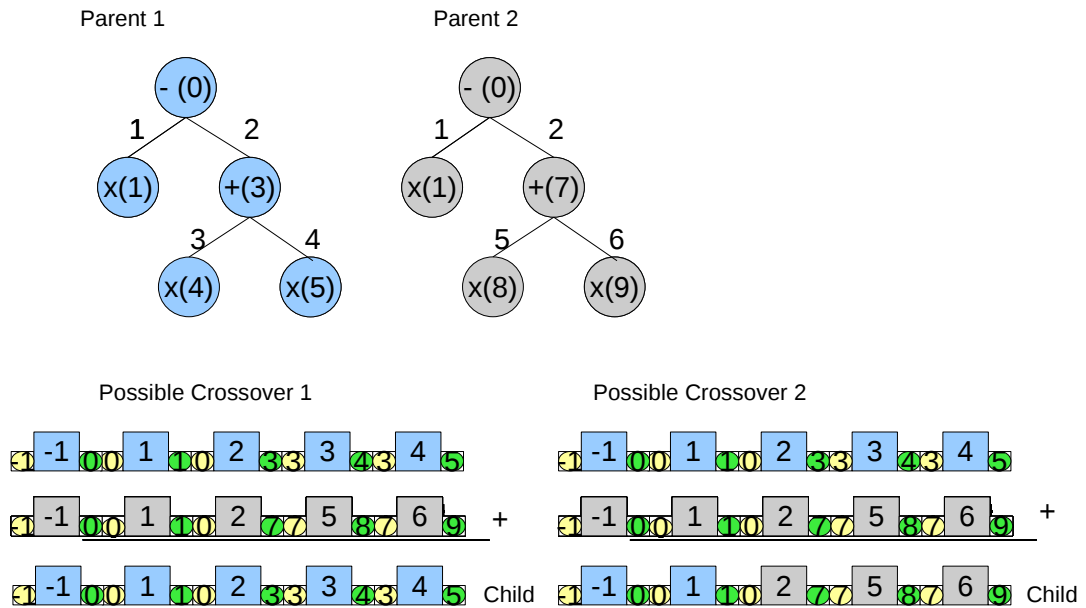


Figure 3.4: **Two possible crossover scenarios.** In the first crossover matching connection 2 is inherited from parent 1 thus connections 5 and 6 are deleted from parent 2’s list since they hang from the rejected connection 2 of parent 2 and also are not common between parent 1 and 2.

### 3.3 Visualizing Change in Tree Structures

Historical markings not only offer an effective way of comparing structures and determining ancestry but also offer an insight into the change in structures of trees. To aid of our understanding of the population dynamics we combine historical markings with a method presented in [DHW05] for visualizing tree topologies. The method shown here provides an algorithm for creating highly compact and scalable “fisheye” tree representations. Using this mechanism they are able to depict trees of large depths as well as summaries of topologies encountered through evolution. We make use of only one aspect of this algorithm and that

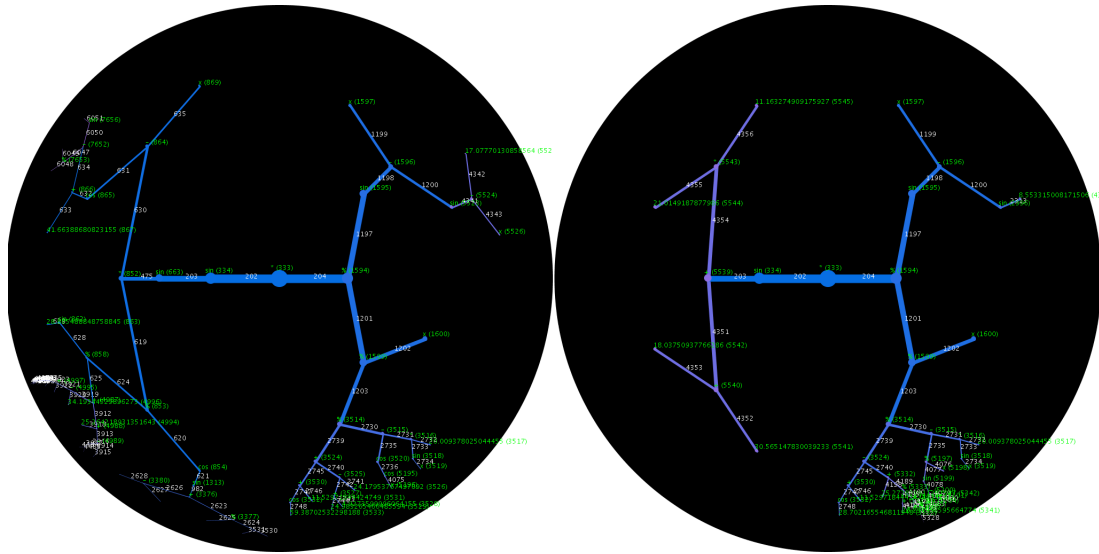


Figure 3.5: **Two example tree visualizations.** Bluer colors reflect older node and branches.

is for representing trees individually. To this we add the visualization of historical markings so that not only are changes in structure observable, but also how old these changes are and in what part of the tree they occurred. Branches and nodes are colored according to their historical markings as seen in figure 3.3. Using this representation one can, for example, by looking at the individuals in the final population understand which root structures survived that were present in the initial population.

## CHAPTER 4

### EXPERIMENTAL ANALYSIS OF NEAT-GP

The purpose of these experiments is to explore the dynamics of a genetic program algorithm that implements NEAT concepts described above and how it compares to traditional GP approaches. We will compare the performance of our approach with that of existing genetic programming implementations. We use the problems of symbolic regression because it has been documented that these have continuous fitness landscapes as opposed to a “needle in a haystack” fitness landscape (i.e. the boolean parity and artificial ant problems) [LP98b]. These type of problems are more suitable for the building approach on which NEAT concepts are based on.

In these experiments we seek to answer the following questions about NEAT-GP regarding **performance** and **bloat**:

1. How many times did it find the solution and how many evaluations does it require?
2. If a solution is not found, how well does it approximate the solution?
3. How much bloat is there in NEAT-GP in terms of size and depth of average population, and size and depth of best individuals?

## 4.1 Experimental Setup

The goal of this experiment is to gather a general idea of the dynamics of NEAT-GP as compared to a standard GP algorithm. We use 2 data sets of 40 points each obtained from the stock market Industrial Dow Jones high values (figure 4.1) and two standard regression equations. Differently than data generated from known equations, real data contains errors and noise. The difficulty of the curves corresponding to the data sets chosen varies from almost linear to having several maximas and minimas. The equations regressed are  $x^5 - 2x^3 + x$  and  $x^6 - 2x^4 + x^2$  which are widely mentioned in the literature. Regression tasks are usually desired for problems where an equation and the characteristics of the problems are unknown. . The function set consists of SIN, COS, \*, %, -, + and the terminal set contains the variable x and ERCs in the range [0,40) for the data sets and [-1,1) for the equation problems.

The parameter settings used for all experiments are shown in tables 4.1 and 4.1. For both algorithms tournament selection of size 7 was used when choosing what individuals to mate. In the case NEAT-GP, tournaments were intra specie only. There was a 0.9 probability of crossover in NEAT-GP and a 0.1 probability of reproductions. Mutation is performed after crossover with the probabilities mentioned in table 4.1. The root, terminals and nonterminals were mutated with the same probability differently from GP where the probability of mutating non terminals was higher than of mutating terminals. Depth limiting crossover was used in GP with a standard maximum depth of 17.

Table 4.1: NEAT-GP parameters.

Population Size:	300
Generations:	Data Sets: 2000, Quintic: 500
Init Tree Algorithm:	Ramped Half-n-half
Init Tree Min Depth:	1
Init Tree Max Depth:	2
Init Tree Algorithm Prob:	0.5
Terminals Mutation Prob:	0.5
Nonterminals Mutation Prob:	0.5
Root Mutation Prob:	0.0
Grow Mutation Tree Min Depth:	1
Grow Mutation Tree Max Depth:	5
Xover + Grow Mutation Prob:	0.8
Xover Only Prob:	0.0
Xover + ChangeOneNode Mutation Prob:	0.1
Xover + DeleteOneNode Mutation Prob:	0.0
Grow Mutation Only Prob:	0.0
Reproduction Only Prob:	0.1
Compat Threshold:	1.0
Compat Threshold Inc:	3.0
Target NumSpecies:	15
Remove Bottom Percent:	20

#### 4.1.1 Results

##### 4.1.1.1 Quintic Regression Results and Discussion

The quintic problem is a simple problem often used along with other benchmarks to quantify the effect of new GP techniques on bloat. Figures 4.1.1.2 and 4.1.1.2 show the summaries over all runs of NEAT-GP and standard GP on the quintic regression problem over 50 runs. For this problem a fixed number of generations (500) was run even after a solution was found. NEAT-GP found solutions in 42 of 50 runs and on average later in evolution than regular GP

Table 4.2: GP parameters.

Population Size:	300
Generations:	Data Sets: 2000, Quintic: 500
Init Tree Algorithm:	Ramped Half-n-half
Init Tree Min Depth:	1
Init Tree Max Depth:	5
Init Tree Algorithm Prob:	0.5
Terminals Mutation Prob:	0.1
Nonterminals Mutation Prob:	0.9
Root Mutation Prob:	0.0
Xover Prob:	0.9
Reproduction Only Prob:	0.1

which found 33 solutions. In NEAT-GP the average number of nodes as well as the nodes of the best individuals is much smaller than in regular GP as seen in the generations vs. number of nodes summary. Results for the sextic equation were similar as for the quintic.

#### 4.1.1.2 Stock Market Data Regression Results and Discussion

Figures 4.1.1.2 and 4.1.1.2 show the summaries over 35 runs of NEAT-GP and standard GP regressing data set No. 1. While neither GP and NEAT-GP find an exact match approximation to the data, figure 4.1.1.2 shows that they both approximate the function very closely. In terms of fitness NEAT-GP finds a best solution of 0.68 fitness from all runs which is slightly better than the best solution found by GP of 0.58 fitness found in run 19 as shown in figure 4.1.1.2. Figure 4.1.1.2 shows the summary graph for run 2 where the best structure was found for NEAT-GP and figure 4.1.1.2 shows an equivalent graph for GP.

However, the most significant difference is that NEAT-GP shows fewer number of nodes in

the average population and in the best individual. This we think is one of the benefits of applying the NEAT approach that provides a good exploration and exploitation of small structures for GP. If there are possible fit and simple structures which satisfy the problem it is likely that they will be found. Speciation provides protection for small trees that are not highly fit so that they can be optimized and grown to their full potential. Through this mechanism it is possible to have two or more species with similar fitness but great differences in number of nodes. When trying to find the smallest structures it is important to keep in the population groups of structures of various fitness and topology so that evolution can attempt to optimize them and choose the one which improves more. For example, in figure 4.1.1.2 is depicted a snapshot of one of our runs where we observe how two species (47 and 233) which have similar fitness but significant number of nodes coexist in the population. Although here we see how specie 47 with highest number of nodes improves to beat the best fitness of the smaller specie 233, the individuals in specie 233 were protected for the previous 5 generations that it did not improve its best fitness. This change causes the new dominant individual of the population to be of complete different structure as shown in figure 4.1.1.2.

Note that in NEAT-GP the number of species is initially large almost as large as the population size since there is little similarity between individuals in the randomly created population. As evolution progresses, the number of species settles to the target number desired by dynamically adjusting the threshold of similarity for comparing trees. Determining the target number of species as well as the compatibility threshold parameters is what we perceived as the most challenging element of this algorithm. The value of the thresholds

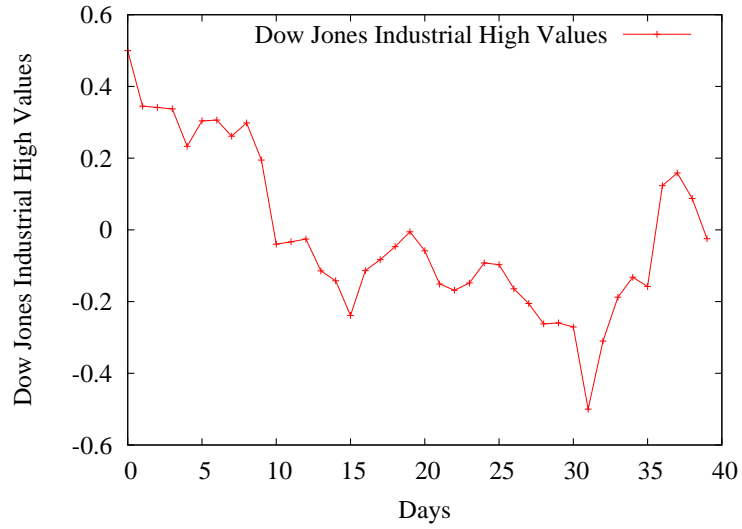
depends highly on how drastic mutation is and how much structure is added or removed to the genomes. In our implementation, because trees have an initial depth between 1 and 2, we chose an initial compatibility threshold of 3 nodes which corresponds to the number of nodes of a full binary tree of depth 2 (if we count the root to be one level).

Data set 2 is a somewhat linear data set and thus poses a simpler problem as observed in summaries shown in figures 4.1.1.2 and 4.1.1.2. The average fitness achieved by NEAT-GP over all runs is higher than achieved by GP. However, differently from the runs with data set 1, here GP finds a higher fit approximation (0.93 fitness) than NEAT-GP (0.80) in one of the runs (figure 4.1.1.2). A closer look into the dynamics of each run shows that although GP is able to greatly exploit specific structures in some runs, it stagnates early in the majority of the runs. NEAT-GP on the other hand, shows consistent improvement in most runs and in cases when there are signs of stagnation these are observed in late generations. Both GP and NEAT-GP show a similar phenomenon where some runs appear to discover a structure that allows for drastic improvement while other runs are more conservative. NEAT-GP however, in absence of such structure seems to continue to optimize the current structures at a reasonable rate.

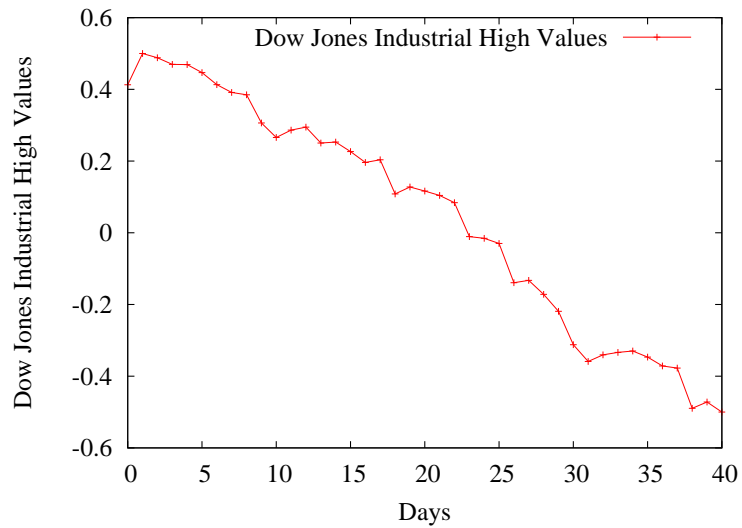
Also, similarly to the results from using set 1, NEAT-GP shows a significant smaller number of nodes in the average population as well as for the best individual. In the case of GP, some runs stand out because of their growth in number of nodes in comparison to the fitness improvement. These runs (an example run shown in figure 4.1.1.2) are responsible for the higher average of nodes over all runs in GP. There were 6 runs of 35 runs which showed this



behavior. NEAT-GP does not show this phenomenon and instead, in all runs the average number of node curve resembles that of the fitness curve.



(a) Data set No. 1. August 06, 2007 to October 1, 2007. Domain and range values have been scaled.



(b) Data set No. 2. March 30, 2007 to May 25, 2007. Domain and range values have been scaled.

Figure 4.1: Data sets 1 and 2 from Dow Jones Industrial High Values.

Gen: 151					
SpecieId	Avg. Fitness	Best Fitness	Num Individuals	Avg. Nodes Per Tree	Avg. Depth Per tree
47	0.198196354	0.252817929	25	159.60000000	19.12000000
233	0.112484246	0.253570169	21	41.142857143	10.285714286
280	0.166743597	0.235744059	24	136.416666667	21.25000000
300	0.053325513	0.066246934	9	121.111111111	14.555555556
315	0.16606071	0.195427775	21	123.380952381	16.142857143
Gen: 152					
SpecieId	Avg. Fitness	Best Fitness	Num Individuals	Avg. Nodes Per Tree	Avg. Depth Per tree
47	0.205120717	0.253696203	29	156.862068966	18.862068966
233	0.128440085	0.253570169	17	34.352941176	9.529411765
280	0.164930425	0.235744059	22	136.545454545	20.818181818
300	0.044402998	0.066246934	8	112.375000000	12.375000000
315	0.144563995	0.217724368	24	117.458333333	16.291666667

Figure 4.2: **Example snapshot of species in two generations.** Species with similar fitness but which differ in structures and the total number of nodes coexist in the population.

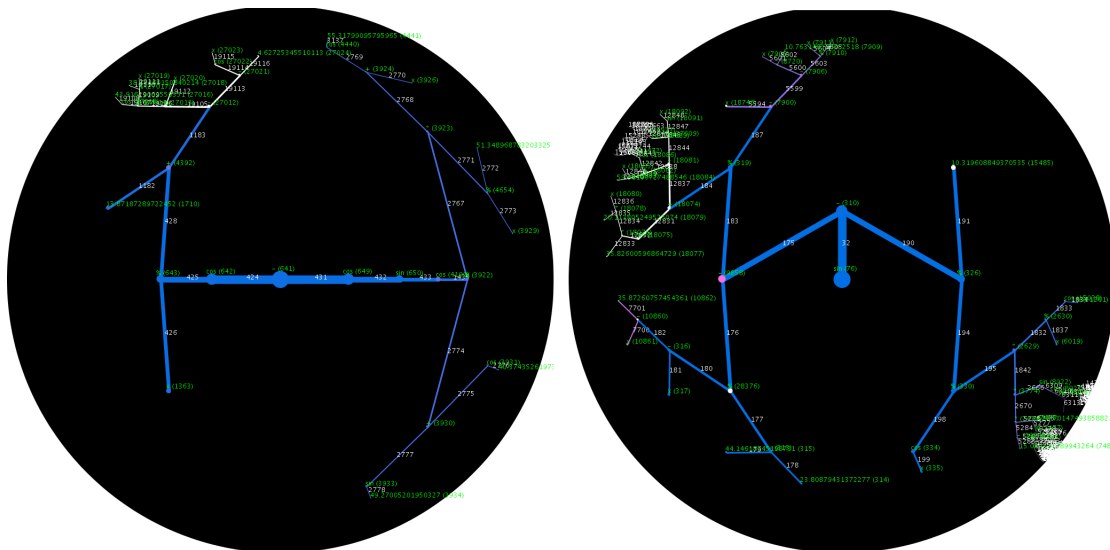


Figure 4.3: **Example of two trees with similar fitness but different structures.** The tree on the right belongs to a specie that beat the specie on the left.

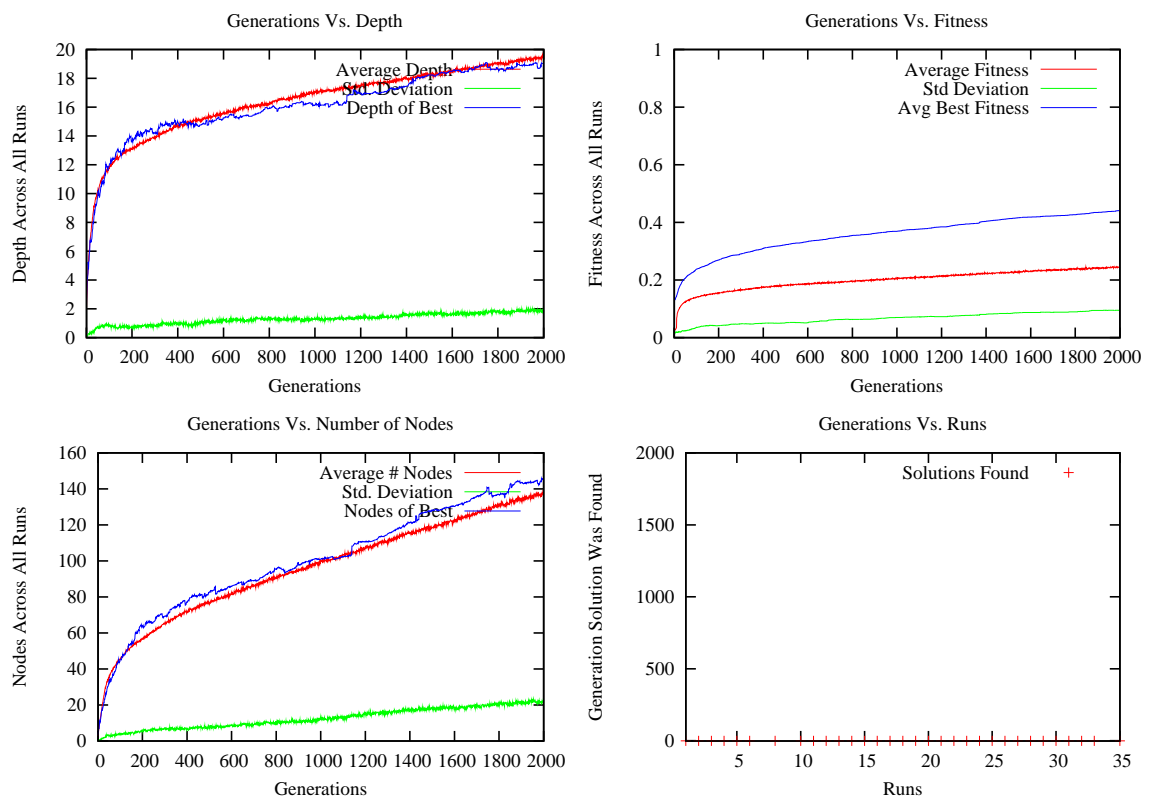


Figure 4.4: **Run summary of experiment 1 using NEAT-GP to approximate data set No. 1.** Highest fitness corresponds to a value of 1. No exact solution was found thus the bottom right diagram shows generations where solutions were found as -1.

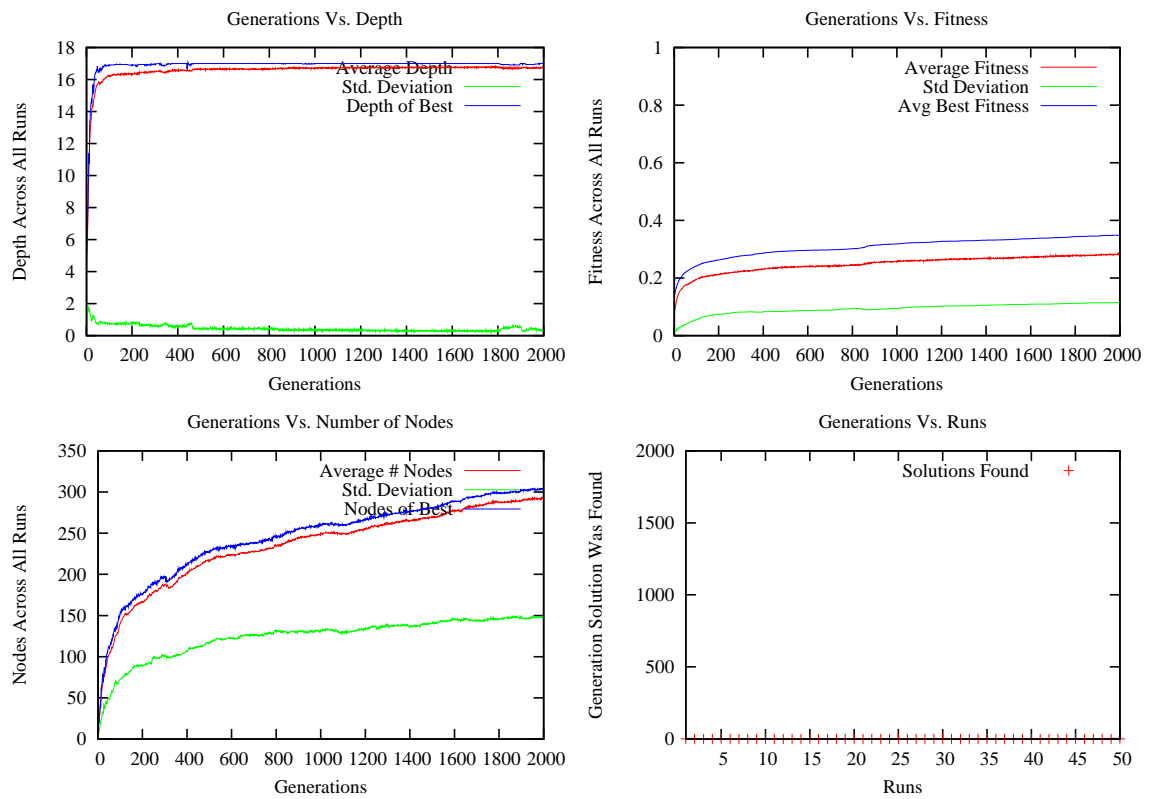


Figure 4.5: **Run summary of experiment 1 using regular GP to approximate data set No. 1.** Highest fitness corresponds to a value of 1. No exact solution was found thus the bottom right diagram shows generations where solutions were found as -1.

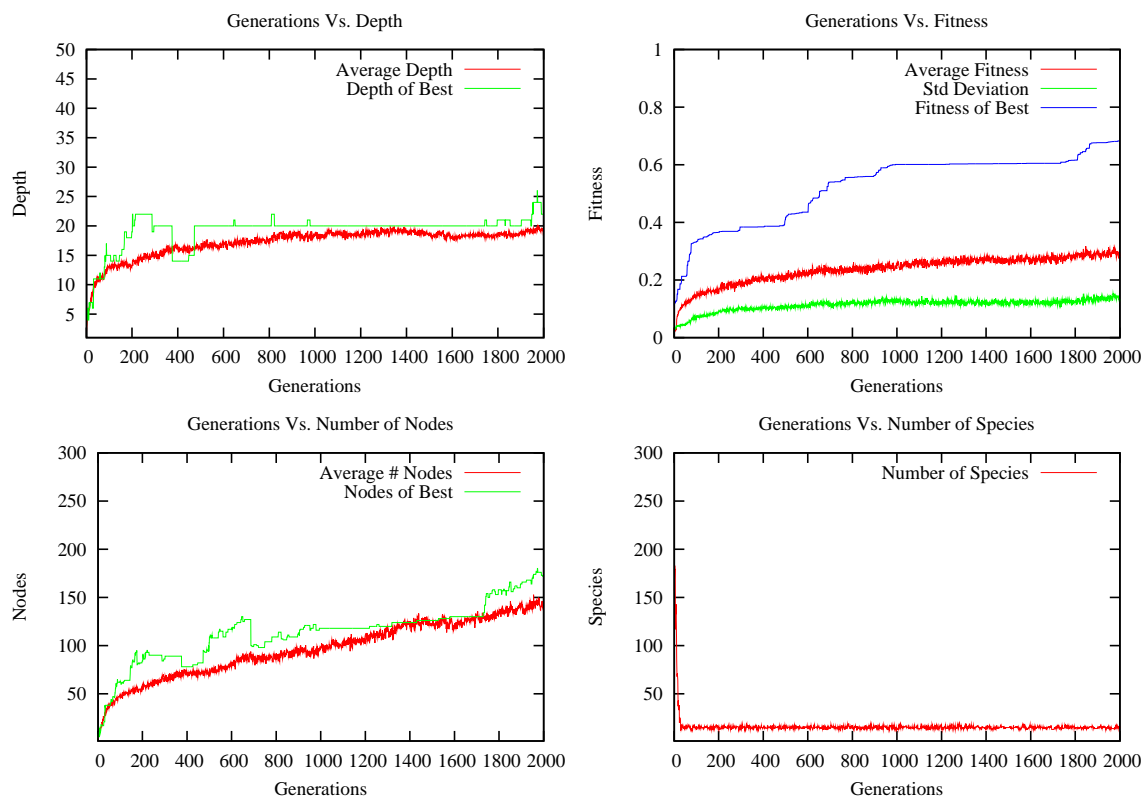


Figure 4.6: **Summary of run 2 where the best structure was found in experiment 1 using NEAT-GP to approximate data set No. 1. This individual has a fitness of 0.684 and was found in generation 1998.**

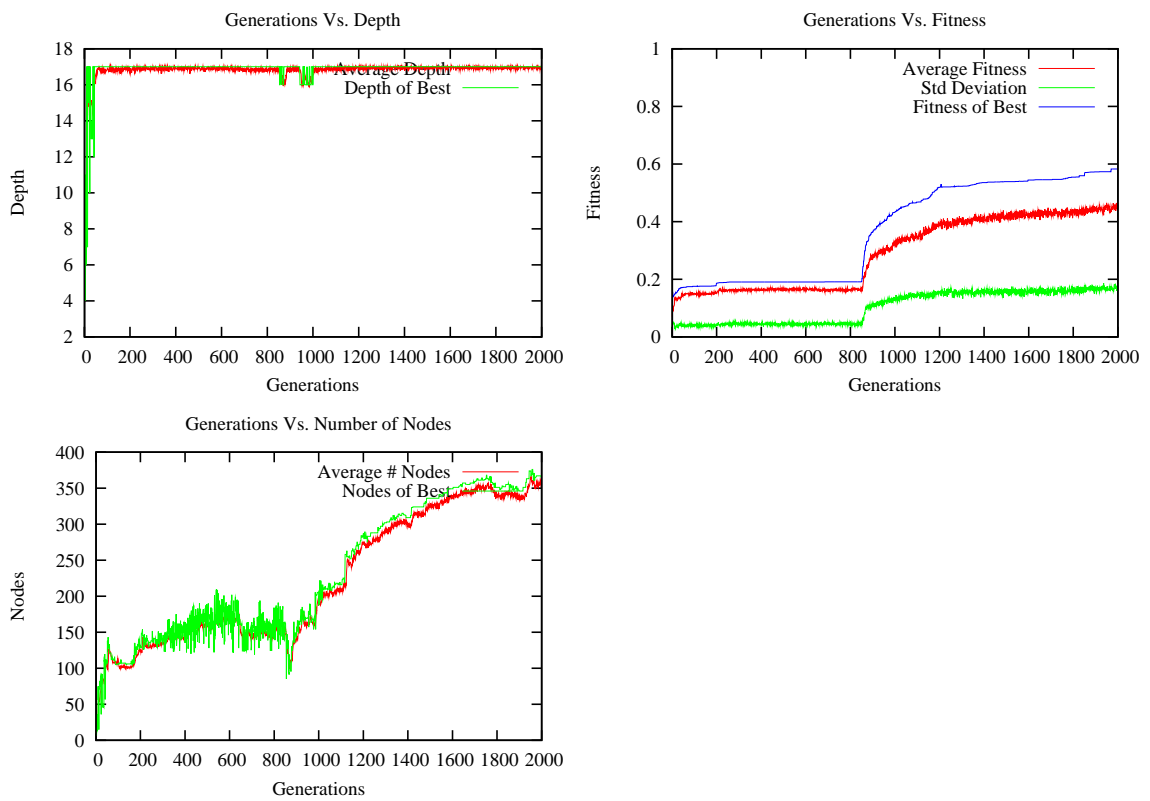


Figure 4.7: Summary of run 19 where the best structure was found in experiment 1 using GP to approximate data set No. 1. This individual has a fitness of 0.58.

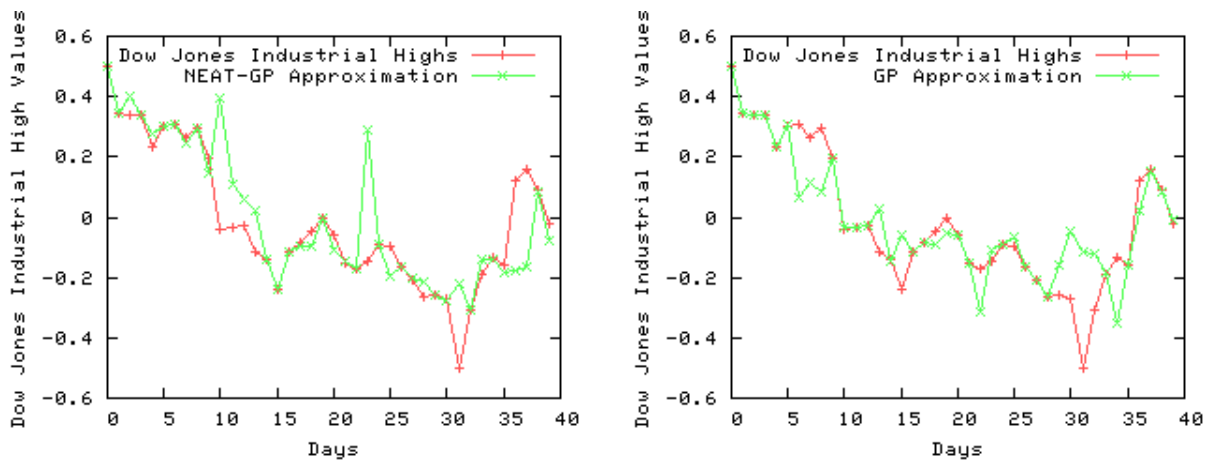


Figure 4.8: Experiment 1: Best approximation found of data set No. 1 using standard NEAT-GP (left) and standard GP (right).



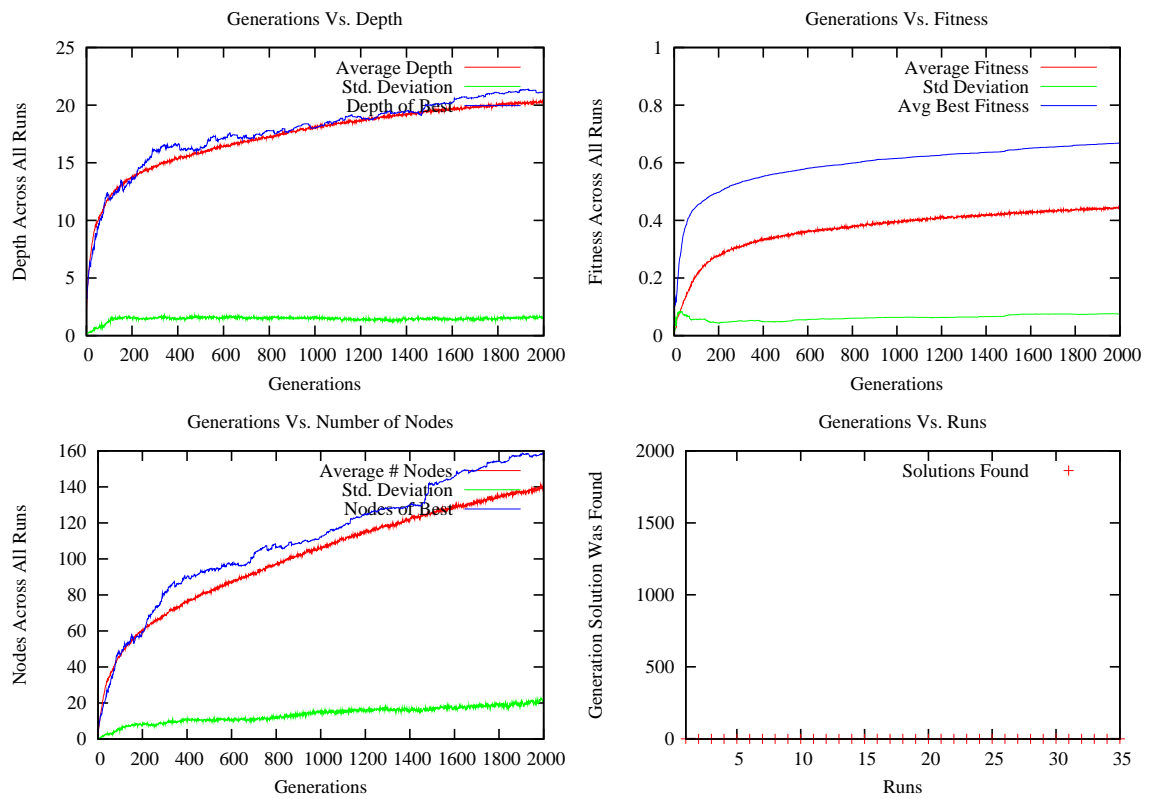


Figure 4.9: **Run summary of experiment 1 using NEAT-GP to approximate data set No. 2.** Highest fitness corresponds to a value of 1. No exact solution was found thus the bottom right diagram shows generations where solutions were found as -1.

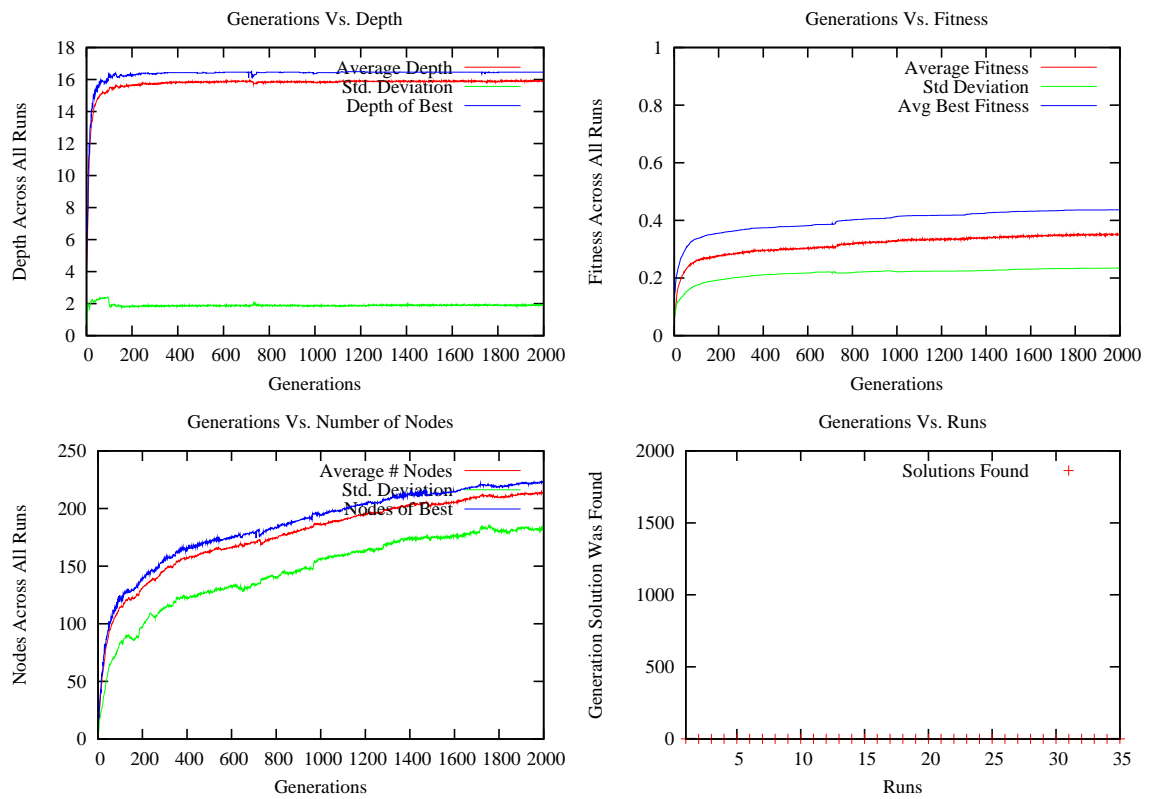


Figure 4.10: **Run summary of experiment 1 using regular GP to approximate data set No. 2.** Highest fitness corresponds to a value of 1. No exact solution was found thus the bottom right diagram shows generations where solutions were found as -1.

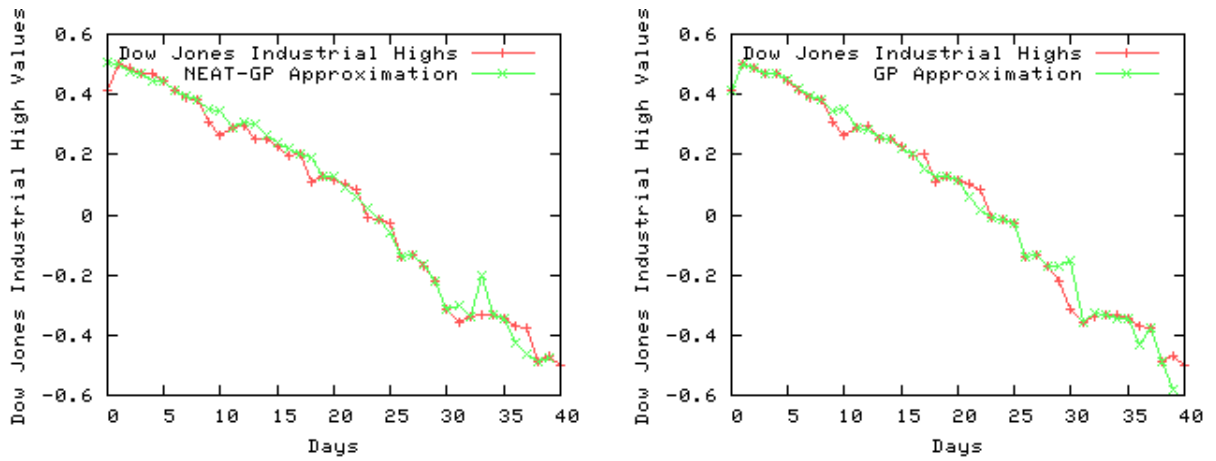


Figure 4.11: Experiment 1: Best approximation found of data set No. 2 using standard NEAT-GP (left) and standard GP (right). Fitness of the individuals encoding these solutions were 0.80 and 0.93 respectively.

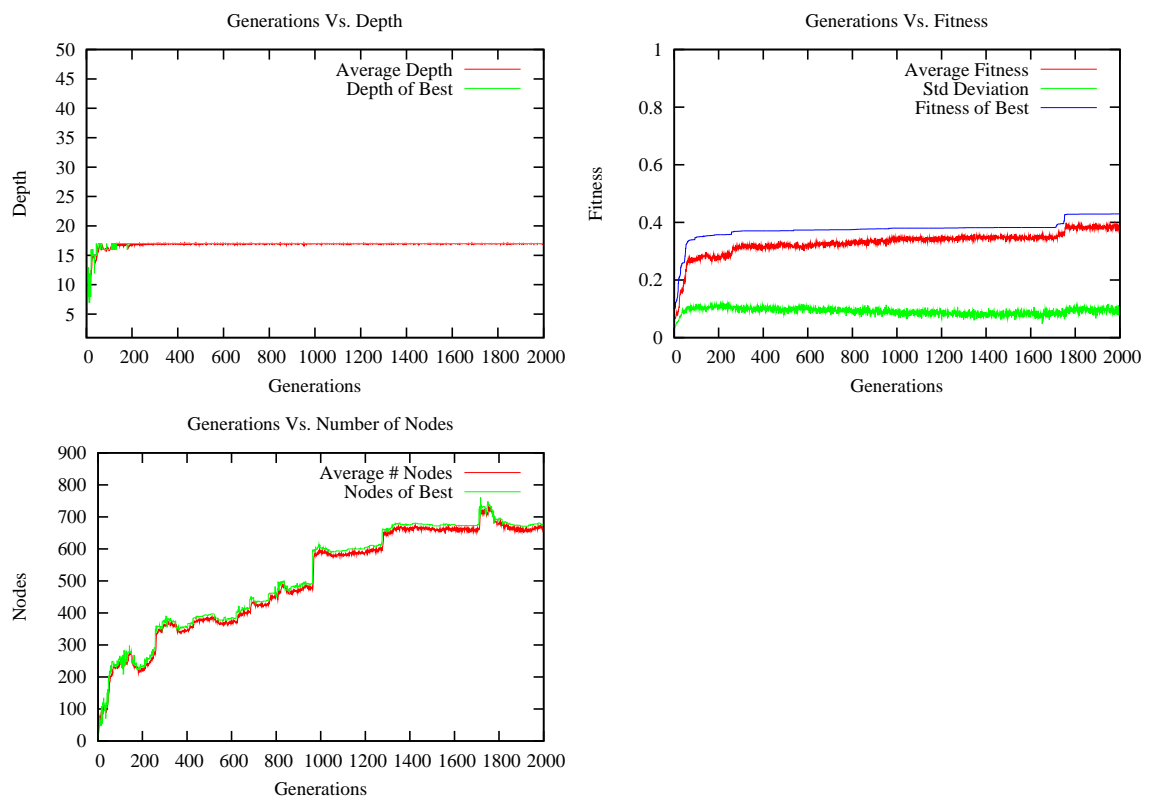


Figure 4.12: **Example run (run 5) from experiment 1 using data set No. 2 and GP.** This run shows a high average number of nodes of the population as well of the best individual which does not match its corresponding slow fitness increase.

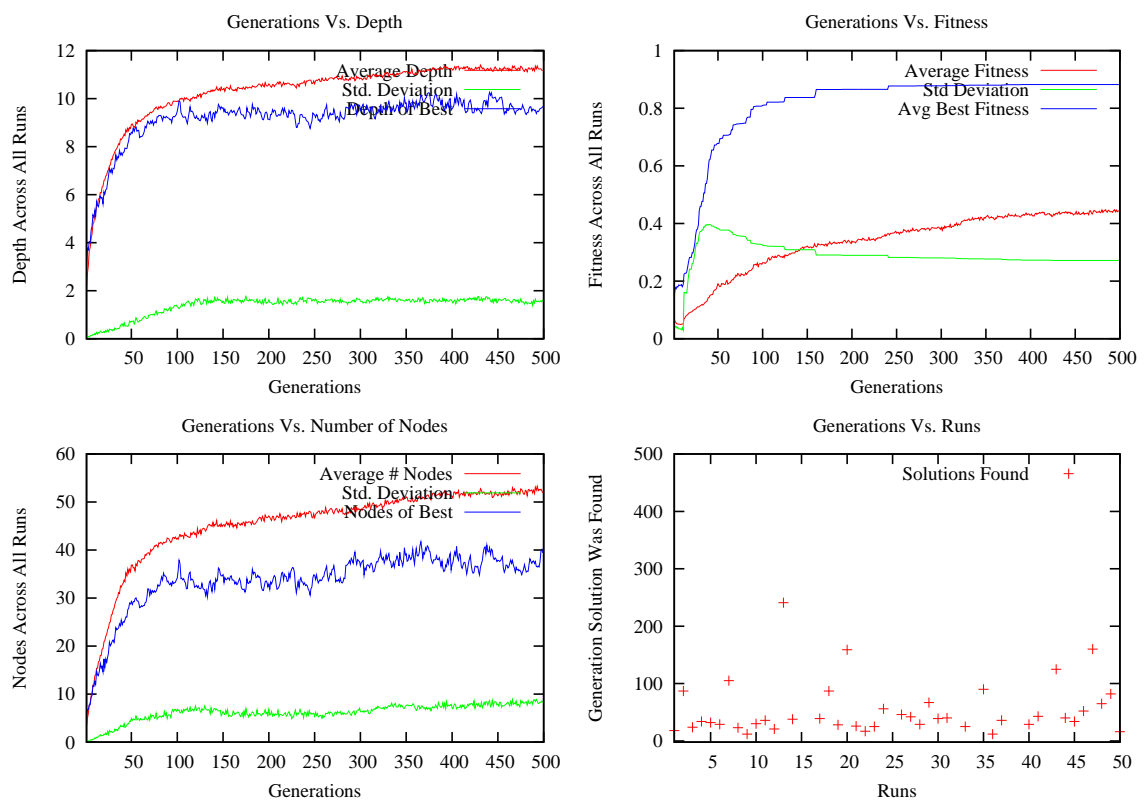


Figure 4.13: Run summary of experiment 1 using NEAT-GP for the symbolic regression problem of the quintic equation. Highest fitness corresponds to a value of 1. All but 8 runs found a solution.

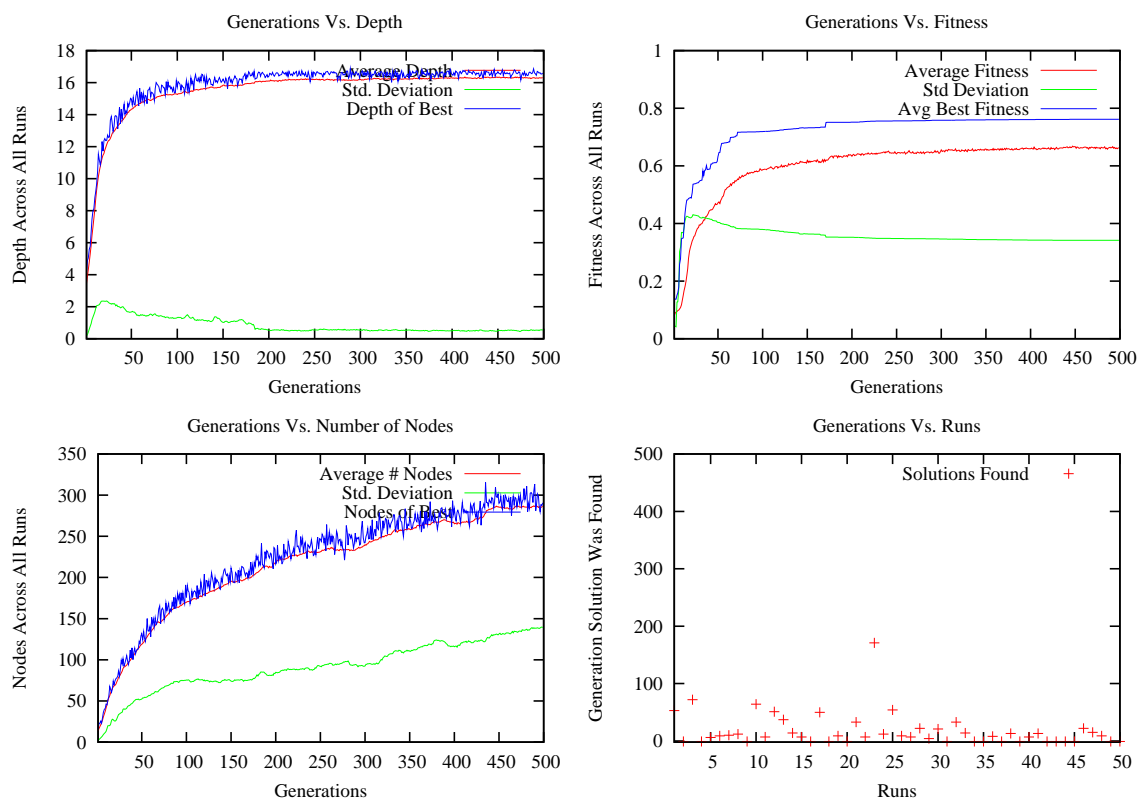


Figure 4.14: Run summary of experiment 1 using GP for the symbolic regression problem of the quintic equation. Highest fitness corresponds to a value of 1. All but 17 runs found a solution.

## CHAPTER 5

### CONCLUSIONS

In this research we set out to understand how concepts from the evolution of neural networks using NEAT could be applied to genetic programming. Most importantly, our goal was to find out how beneficial such an approach would be and its impact in code growth and fitness. We developed mechanisms similar to those in NEAT including a tracking method for trees, mutation and crossover operators which preserved historical information, tree comparison algorithms, and speciation. Although our experiments only focused on symbolic regression problems, they provided useful insights into the dynamics of the algorithm. We found that code growth as measured in number of nodes and depth was significant lower than in regular GP and that the behavior of fitness of NEAT-GP varied slightly among all problems. NEAT-GP showed higher fitness values across all runs and a consistent relationship between the increase in number of nodes and the increase in fitness. This contrasts with the results from the GP runs which have traditionally be characterized by a few runs that reach high fitness at the expense of drastic increase in the number of nodes and many runs that stagnate. We observed how in NEAT-GP groups of structures of similar fitness but difference in complexity coexist in the population through the protection of speciation. This showed to be

a more conservative approach since even when some species had a significant higher fitness than others, they were not allowed to dominate the population. Instead, different structures and fitnesses were maintained through evolution. This allowed NEAT-GP to have a more systematic pace for building trees in which the size of trees was consistently proportional to their fitness. Because there exist other methods which are superior to the standard GP, future work should investigate how NEAT-GP performs in comparison.



## LIST OF REFERENCES

- [BGK04] E. Burke, S. Gustafson, and G. Kendall. “Diversity in genetic programming: An analysis of measures and correlation with fitness.”, 2004.
- [BT94] T. Bickle and L. Thiele. “Genetic Programming and redundancy.” *Genetic Algorithms within the Framework of Evolutionary Computation*, 1994.
- [Che98] Kumar Chellapilla. “A Preliminary Investigation into Evolving Modular Programs without Subtree Crossover.” In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 23–31, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.
- [Dh94] Patrik D’haeseleer. “Context preserving crossover in genetic programming.” In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pp. 256–261, Orlando, Florida, USA, 27-29 1994. IEEE Press.
- [DHW05] Jason M. Daida, Adam M. Hilss, David J. Ward, and Stephen L. Long. “Visualizing Tree Structures in Genetic Programming.” *Genetic Programming and Evolvable Machines*, 6(1):79–110, 2005.
- [DLT03] Jason M. Daida, Hsiaolei Li, Ricky Tang, and Adam M. Hilss. “What Makes a Problem GP-Hard? Validating a Hypothesis of Structural Causes.” In Erick Cant-Paz, James A. Foster, Kalyanmoy Deb, Lawrence Davis, Rajkumar Roy, Una-May O’Reilly, Hans-Georg Beyer, Russell K. Standish, Graham Kendall, Stewart W. Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitchell A. Potter, Alan C. Schultz, Kathryn A. Dowsland, Natasa Jonoska, and Julian F. Miller, editors, *GECCO*, volume 2724 of *Lecture Notes in Computer Science*, pp. 1665–1677. Springer, 2003.
- [FCB99] Frank D. Francone, Markus Conrads, Wolfgang Banzhaf, and Peter Nordin. “Homologous Crossover in Genetic Programming.” In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pp. 1021–1026, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

- [GR87] David E. Goldberg and Jon Richardson. “Genetic algorithms with sharing for multimodal function optimization.” In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pp. 41–49, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.
- [Han03] James V. Hansen. “Genetic Programming Experiments with Standard and Homologous Crossover Methods.” *Genetic Programming and Evolvable Machines*, 4(1):53–66, March 2003.
- [Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.
- [LP98a] W. B. Langdon and R. Poli. “Fitness Causes Bloat: Mutation.” In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391, pp. 37–48, Paris, 14-15 1998. Springer-Verlag.
- [LP98b] W. B. Langdon and R. Poli. “Why “Building Blocks” Don’t Work on Parity Problems.” Technical Report CSRP-98-17, 13 1998.
- [LP02] Sean Luke and Liviu A. Panait. “Lexicographic Parsimony Pressure.” In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 829–836, New York, 9-13 ” jul 2002. Morgan Kaufmann Publishers.
- [LP06] Sean Luke and Liviu Panait. “A Comparison of Bloat Control Methods for Genetic Programming.” *Evolutionary Computation*, 14(3):309 – 344, 2006.
- [Luk01] Sean Luke. “When Short Runs Beat Long Runs.” In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 74–80, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.
- [MH99] Nicholas Freitag McPhee and Nicholas J. Hopper. “Analysis of genetic diversity through population history.” In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pp. 1112–1120, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.
- [NB95] Peter Nordin and Wolfgang Banzhaf. “Complexity Compression and Evolution.” In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International*

- Conference (ICGA95)*, pp. 310–317, Pittsburgh, PA, USA, 15-19 1995. Morgan Kaufmann.
- [PL97] Riccardo Poli and W. B. Langdon. “A New Schema Theory for Genetic Programming with One-point Crossover and Point Mutation.” In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 278–285, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.
- [PL98] Riccardo Poli and William B. Langdon. “On the Search Properties of Different Crossover Operators in Genetic Programming.” In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 293–301, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.
- [RB95] Justinian Rosca and Dana H. Ballard. “Causality in Genetic Programming.”, 15-19 1995.
- [SFD96] Terence Soule, James A. Foster, and John Dickinson. “Code Growth in Genetic Programming.” In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 215–223, Stanford University, CA, USA, 28–31 1996. MIT Press.
- [SM02] Kenneth O. Stanley and Risto Miikkulainen. “Evolving Neural Networks Through Augmenting Topologies.” In *Evolutionary Computation*, volume 10, pp. 99–127. MIT Press, 2002.
- [Tac94] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, Department of Electrical Engineering Systems, USA, 1994.
- [WHR87] J. D. Watson, N. H. Hopkins, J. W. Roberts, and A. M. Wiener. *Molecular Biology of the Gene*. The Benjamin/Cummings Publishing Company, 1987.