Jason M. Daida, Robert R. Bertram, John A. Polito 2, and Stephen A. Stanhope

This chapter addresses the question "what is a building block in genetic programming?" by examining the smallest subtree possible—a single leaf node. The analysis of these subtrees indicates a considerably more complex portrait of what exactly is meant by a building block in GP than what has traditionally been considered.

10.1 Introduction

What is a building block in genetic programming (GP)? Intuitively, we might answer simple pieces of code, subprograms, that GP uses to build more complex programs. Intuitively, too, that idea resonates with some theoretical developments in genetic algorithms. Some have taken building blocks as a given and have built algorithms and systems to enhance their production (e.g., [Iba and de Garis 1996; Rosca and Ballard 1996; Ito et al. 1998]). However, one would be hard pressed to describe exactly what constitutes a building block. One would be even harder pressed to show what they are analytically (i.e., what are the salient mechanisms, processes, and mathematics that describe the creation, propagation, and use of a building block). Answers to those questions would likely be found at the core of understanding the dynamics associated with GP. Unfortunately, as researchers have found, such answers have not been easily forthcoming.

Towards this end, an increasing amount of basic research has focused on addressing the question "what is a building block?" This includes work in GP theory, including [Altenberg 1994; O'Reilly and Oppacher 1995; Whigham 1995; Poli and Langdon 1997; Rosca 1997; Banzhaf et al. 1998]. Of these, [O'Reilly and Oppacher 1995] stands out because they were one of the first who applied the corresponding theory from genetic algorithms to genetic programs. They found that the resulting building block definition was insufficient in describing the kind of dynamics that occur in GP. There is also work, mostly empirical, that either focuses or speculates on the nature of building blocks. These include [Koza 1992; Angeline 1994; Haynes 1997; Poli and Langdon 1997; Langdon and Poli 1998; Luke and Spector 1998; Poli et al. 1998]. Alternatives to what conventional wisdom would dictate about building blocks also appear in [Tackett 1994; Mulloy et al. 1996; Punch et al. 1996; Soule et al. 1996; Angeline 1997; Fuchs 1998].

10.1.1 Current Usages and Definitions

At face-value, the term *building blocks* refers to a conceptually simple definition: simple components out of which more complex things can be made. In evolutionary computation, the term first gained widespread usage in describing the dynamics that underlie genetic

algorithms (e.g., see [Holland 1975,1992; Goldberg 1989).[1] At the outset of GP, Koza (1992) suggested that GP uses building blocks in a similar fashion. Early anecdotal information seemed to bear this out, as a few have reported seeing repeated code in their results that have been highly suggestive of building blocks (e.g., [Tackett 1993]).

However, subsequent research has yielded several meanings of the term *building block*, if only to rigorously define it. A building block of a GP tree has been defined to be a subtree of a solution tree [Iba and de Garis 1996; Ito et al. 1998]; blocks of code [Altenberg 1994]; and a rooted subtree [Rosca 1997]. The dynamics of building blocks, especially concerning how they are created and propagated, has been a subject of contention [Altenberg 1994; O'Reilly and Oppacher 1995; Rosca and Ballard 1996; Haynes 1997; Poli and Langdon 1997]. Part of this contention, however, is driven by what exactly constitutes a building block. We would also maintain that what exactly constitutes a building block has been driven, in part, by the metaphors that have been applied to explain it.

Of these metaphors, two stand out: genotype and phenotype. Many in the field would agree that there is a distinction between the two and that both have origins in the biological sciences. However, what exactly in evolutionary computation maps to genotype and phenotype is open to debate.

Genotype is often equated with structures. Depending on one's point of view, genotype is akin to information carried within bit strings or parse trees. Alternatively, genotype is that which underlies a single trait or a set of traits [Bäck and Fogel 1997]. There is probably little disagreement that the computational genotype is analogous to a biological genotype.

What is meant by phenotype is less than clear. A common definition is that a phenotype is the behavioral expression of the genotype in a specific environment [Bäck and Fogel 1997]. Some researchers simply define phenotype as observed behaviors [Fogel 1992]. Some have gone so far as to define the phenotype as equivalent to vector values, as in those used for fitness scoring [Altenberg 1994]. Others have opted for a more abstract definition by equating phenotype with semantics [Haynes 1997].

The relationships between building blocks, genotype, and phenotype have also been open to debate. The prevailing view is that building blocks are genotypes (structures) and that the mathematical formalism of a building block is a schema. Schemata have long been regarded as similarity templates [Holland 1975, 1992; Goldberg 1989]. With regard to GP,

---

[1]Recently, [Macready and Wolpert 1998] challenges the prevailing formalization of building blocks in genetic algorithms [Holland 1973]. If that paper does hold true, it means that the prevailing formalization may not represent an accurate description of building blocks in genetic algorithms. The outcome is not clear-cut, since genetic algorithms were developed independently from the formalization (i.e., see [Goldberg 1989]). It could mean, for example, that the phenomena of building blocks exists in genetic algorithms, but just not in the way described by the Schema Theorem. The implications of [Macready and Wolpert 1998] are even less clear for GP, since GP theory is not necessarily contingent on theoretical findings in genetic algorithms.

schemata have been further formalized as tree fragments that represent multiple subexpressions (e.g., [O'Reilly and Oppacher 1995; Poli and Langdon 1997]). Schemata have also been formalized to represent single subexpressions (e.g., [Whigham 1995]). Some have formalized schemata as rooted tree fragments (e.g., [Rosca and Ballard 1996]). One alternative view holds that for GP, the genotype and the phenotype are one and the same (i.e., [Nordin et al. 1996]). Another alternative view is that building blocks in GP exist in both the genotype and the phenotype (e.g., [Haynes 1997]).

10.1.2 Objectives

The goal of this chapter is to query the nature of a GP building block by observing single-node subtrees from a fitness-enhancing perspective and a population-building-blocks-dynamics perspective. We have sought to push the extent of what has generally been considered a GP building block—a subtree—by considering the smallest possible subtree—a single leaf node (i.e., a terminal). Of interest to us is how GP discovers and exploits *existing* subtrees. Our interest differs from previous work, which have examined how GP creates and discovers *new* subtrees, then exploits them for solution building. By considering single-node units, we have simplified analysis by setting aside the effects that have been associated with crossover within a building block. As this chapter demonstrates, even if building blocks were somehow indivisible (as are single-node subtrees or automatically defined functions (ADFs)), the dynamics associated with such blocks are far from simple.

While there are a large number of contexts among which to examine a single-node subtree, we have chosen to focus on the contexts concerning whether a single-node subtree is functionally expressed in a GP individual. In our case, this means considering at least a second type of subtree that can control the functional expression of single-node subtrees. For us, this interaction is significant because it addresses the issues of genotype and phenotype.

The crux of this chapter lies in determining whether these single-node subtrees are, in fact, building blocks. Using a simple, conceptual sense of the term, we would say yes, this chapter supports that. On the other hand, the analysis of these subtrees suggests a considerably more complex portrait of what exactly is meant by a building block in GP than what has traditionally been considered.

This chapter consists of six other sections. Section 10.2 discusses the setup and methods covered in this chapter. Sections 10.3 and 10.4 describe the experiments, while Section 10.5 discusses the results of these experiments in the context of the question "what is a building block?" Section 10.6 summarizes our conclusions. The three appendices augment our reasoning behind the case-study selection and type of analyses employed.

10.2 Case Study Description

For our case study, we used an example from symbolic regression and had GP solve for the problem $f(x) = (x + 1)^3$. We analyze the propagation and use of ephemeral random constants (ERCs), a specific type of single-node subtree. The following subsections highlight our reasoning for featuring this problem and describe the setup used in our experiments.

10.2.1 Motivations

One of the earliest, intuitive applications of GP has involved data modeling under the label of symbolic regression (i.e., [Koza 1989]). In [Koza 1992], symbolic regression has been synonymous with function identification, which involves finding a mathematical model that fits a given data set. Closely linked problems have included sequence induction, Boolean concept learning, empirical discovery, and forecasting. Typically, practitioners use GP and symbolic regression in several ways: as a benchmark problem to test GP systems, as a software demonstration or tutorial, and as a means of generating mathematical models for real-world data sets. The latter area includes examples in control systems, bioengineering, biochemistry, and finance.

We specifically chose $f(x) = (x + 1)^3$, in part because there are a few *thousand* approaches to obtain a solution and, in part, because there exist several opportunities in which subsolutions can be reused. This problem has well-known mathematical properties that can be exploited for analysis. For example, GP solutions can be constructed from several different coefficients, two of which can be reused: i.e., "1," "2," and "3." These coefficients appear in the following solutions: $(x + 1)^3$, $(1 + 2x + x^2)(x + 1)$, and $(1 + 3x + 3x^2 + x^3)$. Equivalent solutions can also be constructed with other permutations of addition, subtraction, multiplication, and division. Approximate solutions can also be obtained with rational polynomials. See Appendix A.10.1.

We chose to generate fitness cases on the interval [-1, 0) to introduce some ambiguity to the problem, since that interval exists on just one side of that function's only inflection point. Consequently, both even and odd polynomials could, in theory, be used to approximate a solution.

We had three major reasons to use ERCs to setup GP to solve this problem. First and foremost, ERCs are individually traceable throughout the course of a GP trial (also called a GP *run*; we use the words *run* and *trial* interchangeably). The term *ephemeral* random constant is somewhat of a misnomer, at least for some implementations of GP. In the implementation of GP that we used, an ERC is created just once at population initialization. All ERCs remain—values unchanging—in the population for as long as they are used by at

least one individual. We have specified that at the outset of a GP trial, that ERCs have a uniform probability density function (PDF) (i.e., uniformly distributed between specified values). In this way, usage of certain values, if any, are noted by changes in the probability density function of ERC values. Also at the outset of a GP trial, there are large numbers of ERCs that are generated and used in a population. In our case, this amounts to several thousand unique ERC values (floating point, double precision) being used by every trial—enough to generate ERC statistics at the level of a generation within a GP run.

Second, ERCs have variable worth with respect to solving for $f(x) = (x + 1)^3$. For instance, an ERC can either be "noise" or be a "contributing" value. In effect, GP has to solve not one, but two problems. One problem involves creating a mathematical model such that this model fits the supplied data points. This problem is the one a user specifies. The other problem involves creating error-correcting mechanisms to deal with errant ERC values, as we show in Appendix A.10.2. This other problem is an emergent one that GP needs to address in order to solve for $f(x)$. We can illustrate the latter, emergent problem with the following scenario. Let $f'(x)$ be an individual in a GP population. Furthermore let $f'(x) = f(x) + r$, where r is an ERC with a value of 5. GP can obtain the desired solution $f(x)$ in the next generation by *eliminating r*, i.e., by exchanging r with a subtree that evaluates to zero. GP might also be able to *absorb r* by multiplying that ERC with a subtree that evaluates to zero. In this scenario, either elimination or absorption represent error-correcting mechanisms that deal with errant ERC values.

Third, the valuation of an ERC at any given generation may not be well correlated with the larger context for solving for $f(x)$. Just as GP would need to eliminate "extraneous" ERCs, ERCs on the whole would be propagating and increasing in number. Furthermore, this propagation and increase in number would occur in spite of the different approaches to solving for $f(x)$. What we would need to do in order to demonstrate this is measure the change in ERC PDFs. In a sense, ERC implicit fitness would be measured by counting the total number of ERCs in a population and noting what ERC values were used. If there exists an implicit fitness for ERCs, there would likely exist a pattern in ERC PDFs that would transcend the approaches used by GP to solve for $f(x) = (x + 1)^3$. In Appendix A.10.3, we present a derivation supporting this conjecture.

10.2.2 Method

We have listed three reasons for using ERCs, with the latter two suggesting the following questions:

1. How does GP solve for the data model that fits data points generated by the equation $f(x) = (x + 1)^3$, given that ERCs may both aid or interfere with this process?

2. How do ERCs remain in a population in spite of some ERCs having questionable worth with respect to solving $f(x)$?

These questions suggest two different viewpoints for analysis. One viewpoint is fitness-centric and focuses on how ERCs are used by GP to solve for $f(x)$. The other viewpoint is ERC-centric and focuses on how ERCs are maintained in a population. We examine each of these viewpoints by conducting an experiment. For either viewpoint, we emphasize that we are looking at the same phenomena.

Those phenomena are described by the following experimental setup. Fitness cases were 50 equidistant points generated from the equation $f(x) = (x + 1)^3$ over the interval [-1, 0). Raw fitness score was the sum of absolute error. A hit was defined as being within 0.01 in ordinate of a fitness case: 50 hits total. The stop criterion was when an individual in a population scored 50 hits. Adjusted fitness was the reciprocal of the quantity one plus the raw fitness score.

The terminal set consisted of $\{X, \mathbf{R}\}$. ERCs in $\mathbf{R}$ were generated with a uniform distribution over a specified interval of the form $[-a_{\mathbf{R}}, a_{\mathbf{R}}]$, where $a_{\mathbf{R}}$ is a real number that specifies the range for the ERCs. The function set consisted of $\{+, -, \times, \div\}$, where $\div$ is the protected division operator that returns one if the denominator is exactly zero.

We used lilgp to generate the data.[2] Most of the GP parameters were identical to those mentioned in Chapter 7 [Koza 1992]. Population size = 500; crossover rate = 0.9; replication rate = 0.1; population initialization with ramped half-and-half; initialization depth of 2–6 levels; and fitness-proportionate selection. Other parameter values were maximum generations = 200 and maximum tree depth = 26. (Note: these last two parameters differ from those presented in [Koza 1992], which specifies a maximum number of generations = 51 and a maximum depth = 17. Part of the reason we extended these parameters was to avoid possible effects that occur when GP processes individuals at these limits.)

---

[2]We used a patched version of lilgp v.1.02 [Zongker and Punch 1995], a C implementation of GP that is in use in the research community. The patches came from three sources: Luke, Andersen, and Daida. Luke's patches consist of memory leak fixes, multi-threading bug fixes. His enhancements also include provisions for strong-typing (which we did not use) and population initialization. Andersen's fixes included patches to Luke's population initialization routine, so that population initialization could include integer-valued ERCs. Our patches include modifications to the population initialization routine, so that population initialization could include real-valued ERCs.

Our patches also include a different random number generator (RNG). The effect of an RNG on empirical results has been noted in [Koza 1992; Daida, Ross et al. 1997]. Of concern has been that the empirical results obtained are possibly biased: differences between theoretical and empirical results could exist not because of genuine discrepancies between either, but because of idiosyncrasies corresponding to a particular random number generator. Because of concerns with the generator used in lilgp (see [Daida, Ross et al. 1997]), we used the Mersenne Twister [Matsumoto and Nishimura 1997; Matsumoto and Nishimura 1998], a recent variant of TT800. This particular generator is fast, has immense periodicity ($2^{19,937}$ -1, as opposed to 2^{31} -2 for some generators), and has excellent theoretical support for its use. We note, too, that we used lilgp in single-thread mode (as opposed to using multi-thread) because of possible concerns in parallelizing RNGs. (See [Hellekalek 1997; Hellekalek 1998] for recent accounts on choosing and using an appropriate RNG.) *cont.*

10.3 Fitness-Centric Experiment

The first experiment addressed the following question: how does GP solve for the data model that fits data points generated by the equation $f(x) = (x + 1)^3$, given that ERCs may both aid or interfere with this process?

To understand ERCs from this fitness-centric viewpoint, we changed the range a_R of the ERCs. We used three values of a_R: 1, 10, 100. We also ran one control with no ERCs. Four data sets were collected: Control (no ERCs), Unity (ERC: [-1, 1]); Ten (ERC: [-10, 10]); Hundred (ERC: [-100, 100]). Each data set consisted of 600 trials for a total of 2400 runs for this experiment. (We note that increasing a_R does *not* increase the size of the combinatorial search space for GP. While it is true that increasing a_R is likely to increase the absolute value apportioned to each ERC, the total number of ERCs observed in any given population remains statistically constant. See Appendix A.10.3.)

What we were looking for was the effect of varying ERC content on GP's ability to solve for $f(x)$. If ERCs are building blocks (which presupposes "useful" content), we should note an effect by changing ERC values. Towards this end, we were interested in the gross characteristics of the best-of-trial individuals: e.g., size (number of nodes), generation in which a best-of-trial individual is found, depth, and adjusted fitness.

10.3.1 Fitness-Centric Results

Figure 10.1 summarizes and discusses the results from the following data sets: Control, Unity, Ten, and Hundred. Each plot shows 600 points, with each point corresponding to a best-of-trial individual. Columns are arranged by data set.

In creating the plots for the second and third rows, we added a small amount of uniform random noise to both (x, y) coordinates of each point. We did this for visualization only. The quantities corresponding to node count, depth, and generation are integer values—because of this, a single dot could correspond to many data points. The noise was added to displace points visually away from each other. That technique was not repeated for the first row, if only because adjusted fitness is a real-, not integer-valued quantity.

Footnote 2 cont. We also installed non-invasive data taps to collect population snapshots of a run. One of these taps is a modified version of a checkpoint file, which saves the entire state of a GP system at some intermediate step. This tap allows for capturing populations at some arbitrary interval prior to the specified last generation. These population dump files contain only an ERC listing, plus a human-readable version of every individual in that population. This tap also allows for us to capture the initial population, which lilgp does not do when checkpointing.

This patched version of lilgp v.1.02 served as the system for most of the trials that we discuss in this chapter. For the remaining runs, we generated one additional version (called version No-Null). In this version, we installed a problem specific patch in the population generation portion of lilgp. In Section 10.4, we note that a prevalent and pivotal structure was a null of the form `( - X X)`. This patch was designed to knock out that null by replacing the last `X` in this structure with an ERC.

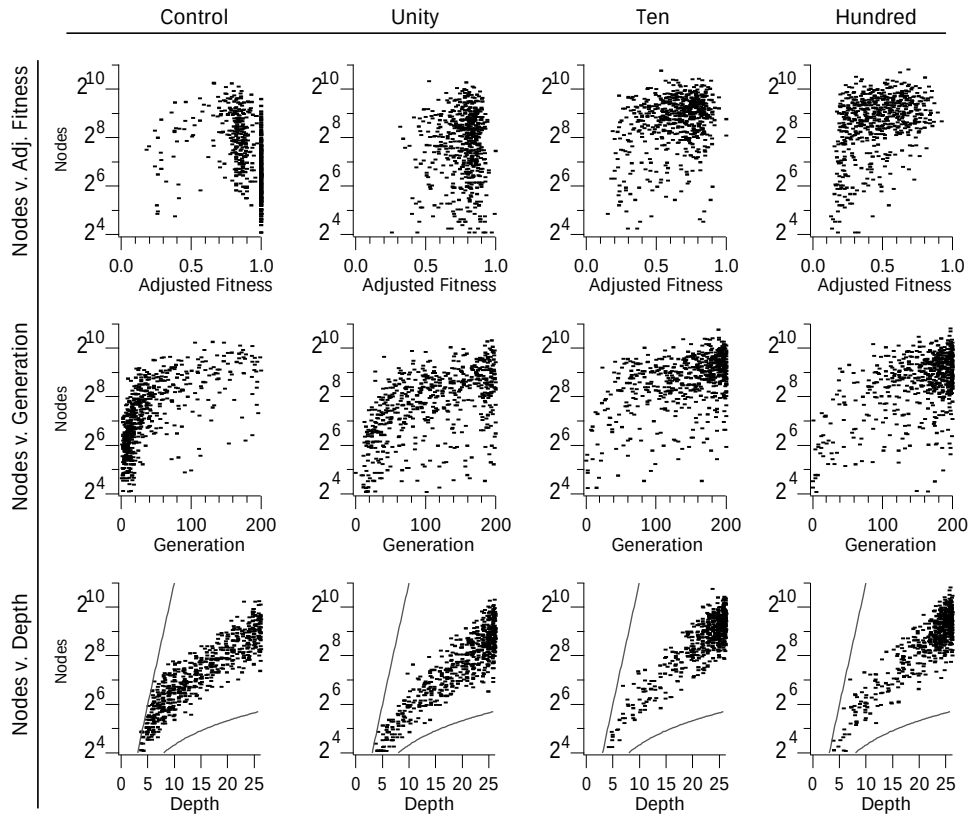

Figure 10.1
Best-of-trial results of fitness-centric experiment. Each column summarizes a data set, where each data set consisted of 600 trials. This figure shows the effect of increasing ERC values on the size and shape of best-of-trial individuals.

Figure 10.1 First Row shows the effect of ERC range on node count versus adjusted fitness. There are what appear to be four groupings. These groupings are indicated roughly as follows: a line interval between 2^5 and 2^9 nodes with adjusted fitness 1.0; a vertical cloud that exists in the interval between 2^5 and 2^{10} nodes with an adjusted fitness of nominally 0.85; a horizontal cloud that exists in the interval of 0.2 and 0.9 in adjusted fitness with a node count of nominally 2^9; and a steeply inclined cloud in the interval between 2^4 and 2^{10} nodes with adjusted fitness of nominally 0.15.

Figure 10.1 Second Row shows the effect of ERC range on node count versus the generation in which the best-of-trial individual was identified. Note that individuals that occur near generation 0 are concise and have likely required less computational effort to generate

than those solutions that occur near generation 200. The distribution suggests two groupings, one near generation 0 and another near the maximum number of generations (200).

Figure 10.1 Third Row shows the effect of ERC range on node count versus the depth of the best-of-trial individuals. The lines indicate the upper and lower bounds for the number of nodes that can be present in a tree for a certain depth.[3] There appears to have been only one grouping that occurs near the maximum depth specification.

10.3.2 Fitness-Centric Discussion

ERCs can have a significant effect on whether GP can solve for $f(x)$. Essentially, the greater the ERC range beyond $a_R = 1$, the more difficulty GP encountered. The easiest task for GP was the Control case (no usage of ERCs), with 283 individuals (out of 600 possible) having perfect 1.0 adjusted fitness scores. The next easiest was the Unity case, except there was only one individual having a perfect adjusted fitness score.[4]

The overall effect of increasing the range of ERCs used in our problem was detrimental. The detrimental effect increased with increasing a_R, even though there were no statistical differences in the total number of ERCs that were initially allocated.

The idea of increasing difficulty with increasing a_R is not altogether surprising. We note, however, that the reason for these phenomena was *not* because there were many more ERCs from which to choose. Rather, the phenomena of increasing difficulty may have occurred because there can be an increase in fitness penalty for choosing an increasingly "wrong" ERC. As an illustration, let r be an ERC value in a GP individual of the form $(x + r)^3$. The consequences for having r picked from the interval [-1, 1] differ markedly for having r picked from the interval [-100, 100]: on the whole, fitness scores would be lower for [-100, 100]. (Alternately, the phenomena of increasing difficulty may have also occurred because there are fewer numbers of potentially useful ERCs from which to use.) For that reason, a significant fraction of ERCs in Ten and Hundred would likely be "noise." On the other hand, ERCs in Unity could serve either as "noise" or as "contributing" value. A manual examination of 50-hit individuals in Unity (numbering 219 best-of-trial individuals) verified that many of those individuals had ERCs integrated into the solutions represented by those individuals.

---

[3]The upper bound represents trees that are completely filled (i.e., binary trees with no vacancies). The lower bound represents sparse trees, where for the most part, each depth consists of one operator and one terminal.

[4]Adjusted fitness and hits served two different purposes in our experiments. Adjusted fitness was used in determining fitness-proportionate selection. Hits were used as a rough indicator of solution quality—50-hits means that an individual has met this indicator perfectly. Note that a 50-hits individual does not necessarily correspond to a perfect-adjusted-score individual, which is $f(x)$ exactly and has an adjusted score of exactly 1.0. A 50-hits individual generally had an adjusted score in the range of 0.8 – 1.0. Consequently, although there was only one perfect-adjusted-score individual in Unity, it turned out that 219 trials (out of 600) had 50-hits (i.e., a third of the trials produced a "reasonable" individual). Likewise for Control, there were 502 trials (out of 600) with 50-hits.

We selected the Unity data set for further examination, in part because the effect of ERCs for that data set was *both* deleterious and positive. We also note that in terms of the groupings shown in Figure 10.1, the Unity data set seemed to represent a transitional "snapshot" between Control and the remaining data sets.[5]

10.4 ERC-Centric Experiment

The second experiment addressed the following question: how do ERCs remain in a population in spite of some ERCs having questionable worth with respect to solving $f(x)$?

To understand ERCs from this ERC-centric viewpoint, we manipulated the contexts in which ERCs can appear. (This is in contrast to the fitness-centric experiment, which involved manipulating ERC *contents*, not their *contexts*.) As we mentioned earlier, while there are a large number of contexts among which to examine ERCs, we have chosen to focus on just a few. In particular, we have been interested in whether an ERC is expressed in a GP individual. Towards that end, we have identified a key structure as `(- X X)`, which we have called the null structure.

We note that there are many other null structures that are possible (i.e., those that map to exactly zero everywhere in x) in the course of a GP run. However, it is `(- X X)` that represents the highest probability structure to occur either at population initialization or later. (Note that this even exceeds the probability of obtaining an ERC value of an exact zero by several orders of magnitude.) See Appendix A.10.2. For that reason, we use the terms *null*, *null structure*, and `(- X X)` interchangeably in subsequent sections of this chapter. Note that a null structure is similar to an approximate zero because it can be used to zero out other subtrees (when used with multiplication or division). A null structure is distinguished from an approximate zero because only it can be used with protected division to create an exact one, which happens to be a coefficient for $f(x) = (x + 1)^3$.

Null structures introduce the most pronounced context shift possible for an ERC. Depending on where an ERC occurs with respect to a null structure, an ERC may ultimately be incorporated to form a solution coefficient or it may ultimately be voided.

We subsequently generated a companion data set, Unity No-Null, with a No-Null version of the kernel. In this version of kernel, all occurrences of the null structure `(- X X)` were removed at population initialization and replaced with `(- X r)`, where $\mathtt{r} \in \mathbf{R}$. Unity

[5]Of particular concern has been the groupings near limits of specification (e.g., maximum generations = 200 or maximum depth of trees = 26). These groupings are essentially artifacts. However, other works (e.g., [McPhee and Miller 1995; Banzhaf, Nordin et al. 1998]) suggest that different processes (e.g., compression) may predominate in regions suggested by these groupings. For instance, Banzhaf, Nordin et al. (1998) would contend that dynamics that describe GP at the beginning of a run would differ from the dynamics when GP operates on individuals that are near the specifications for the maximum size.

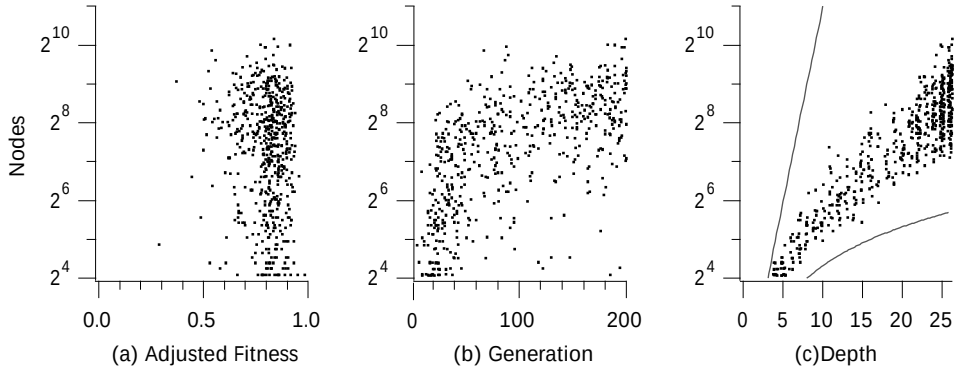

Figure 10.2
Best-of-trial results of ERC-centric experiment. These three scatterplots depict results from Unity No-Null, the companion data set to Unity (shown in the second column in Figure 10.1). This figure shows the effect of removing `(-X X)` from population initialization.

No-Null consisted of 600 trials and ERCs in the range of [-1, 1]. Furthermore, we reran both the Unity and Unity No-Null data sets and took a snapshot of both the initial population and the population in which the best individual was found. The specific interval of which to take these snapshots was subsequently determined *a posteriori*.

Of interest are the gross statistical characteristics concerning ERCs. In particular, if building blocks do exist among ERCs, we should eventually find evidence for that in a nonuniform statistical distribution of ERCs. Furthermore, if the contexts in which ERCs appear are significantly altered, we should find a corresponding change in statistical distribution between Unity and Unity No-Null.

10.4.1 ERC-Centric Results

The results in this section have been divided into two parts. The first-half results overlap Section 10.3 by examining a few fitness-centric characteristics of the Unity No-Null data set, the companion to Unity. We expect differences between Unity and Unity No-Null, if only because we have altered the contexts in which ERCs appear in GP individuals. These differences should subsequently appear at the level of individual.

Figure 10.2 summarizes part of the Unity No-Null data set, the companion to Unity. Figure 10.2a depicts a slight but noticeable fraction of points that have been affected by removing the null structure. The shift in pattern is downwards: in comparison to Unity, the Unity No-Null results show an enhancement in the region from 2^4 to 2^6 nodes and a shift of the cloud from 0.5 to 0.7 in adjusted fitness. Figures 10.2b and 10.2c show slight shifts towards the origin in comparison to their Unity counterparts in Figure 10.1.

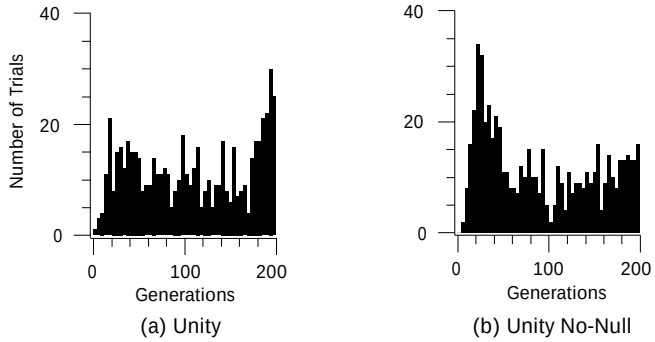

Figure 10.3
Comparison of histogram distributions of generation in which a best-of-trial individual was found. Removing `(- X X)` reduced the computational effort required to produce a best-of-trial individual.

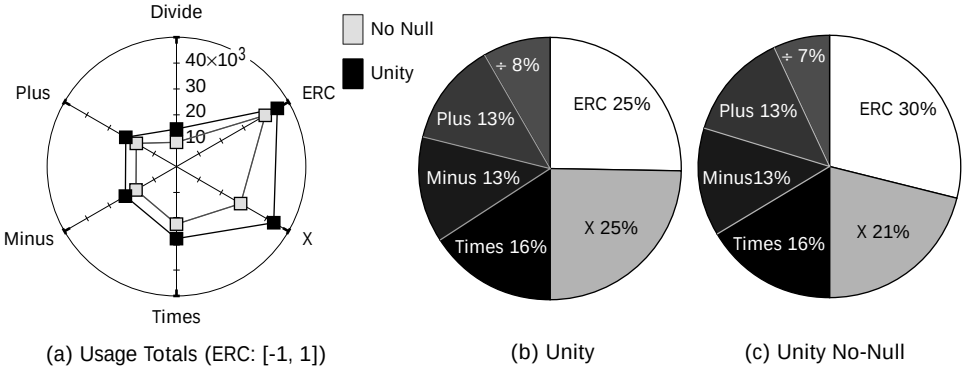

Figure 10.4
Comparison of node-type distributions for best-of-trial individuals from Unity and Unity No-Null data sets. This figure shows the effect of removing `(- X X)` on components that made up best-of-trial individuals.

Figure 10.3 compares histograms from Unity and Unity No-Null. Each histogram depicts the frequency of trials versus the generation in which a best-of-trial individual was identified. This figure shows a slight but noticeable fraction of trials that shifted away from the maximum generation possible (200) in the Unity data set to the peak at 30 generations in the Unity No-Null data set.

In the second-half results, we distinguish between node types and focus the remaining analysis on ERC distributions.

Figure 10.4 shows distributions of all node types for the best-of-trial individuals in both Unity and Unity No-Null data sets. The spider plot at left compares the raw counts of terminals and functions that were used in constructing best-of-trial individuals for Unity and

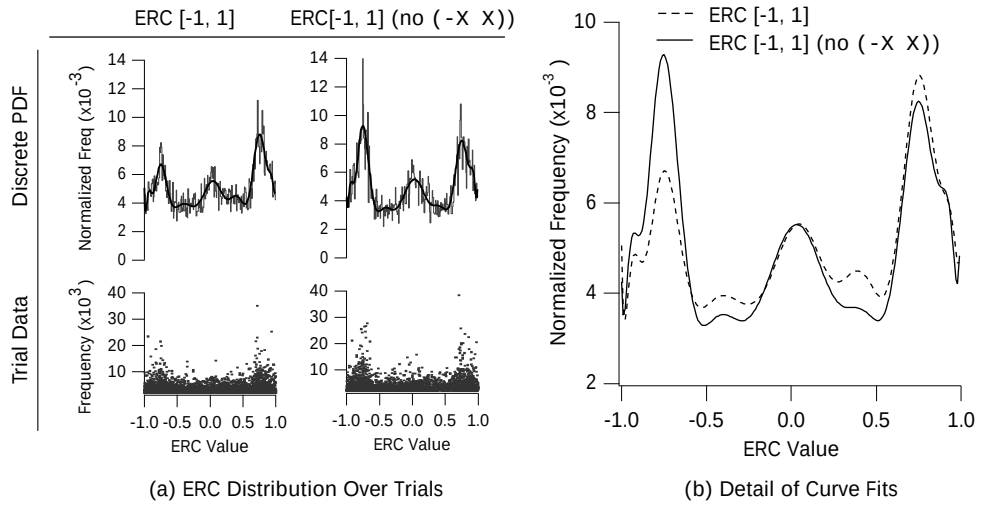

Figure 10.5
Comparison of ERC distributions. This figure shows that ERCs were nonuniformly distributed by the times the best-of-trial individuals were identified. The statistics shown in this figure are for all ERCs in all individuals in every population that a best-of-trial individual was identified (600 populations total). Figure 10.5a shows the intermediate steps taken to generate Figure 10.5b.

Unity No-Null. The hull pattern described by Unity No-Null is not affine with respect to the corresponding pattern for Unity. The pie charts at middle and right have the raw counts normalized. The pie charts in Figure 10.4 show that the ratio of ERCs to X in Unity is lower than the corresponding ratio for Unity No-Null.

Figure 10.5 shows the statistics corresponding to ERC distribution. The left-hand side illustrates our process of data reduction. The bottom row corresponds to the raw data collected from Unity and Unity No-Null. As mentioned in Section 10.2.1, we needed to look at frequencies of ERC values in a population, and not just for best-of-trial individuals. Consequently, each point in the bottom row corresponds to the frequency count of an ERC value as it occurs in a population for a single trial. Each scatterplot subsequently depicts the ERC node counts from 30,000 individuals (500 individuals per population per trial, 600 trials total). The scatterplot for Unity, then, is a summary of 26.0 million ERCs; for Unity No-Null, 23.2 million.

Since scatterplots represent only the ensemble of histogrammic data per trial, we needed to integrate the trial data over all trials to create a normalized distribution: a numerical PDF with discretized bins in 0.01 intervals. The staircase plots in the upper row just above the scatterplots show the numerical PDFs corresponding to Unity and Unity No-Null. To indicate general trends, we overlaid a 20th-order polynomial fit (dark line) on each staircase plot (gray line). The right-hand side of Figure 10.5 is a comparison of those general trends.

10.4.2 ERC-Centric Discussion

The first-half results showed that by altering the low-level contexts in which ERCs appear in GP individuals, we can induce changes at the higher level of individual.

By removing the null structure `( - X X)` from the initial population, we have essentially removed a means by which ERCs can "hide" in the introns (i.e., unexpressed code) of an individual (i.e., either through multiplication or protected division of ERC-bearing subtrees with the null structure). We suspect that these otherwise hidden ERC values would then appear as "contributing" values, which would subsequently affect the fitness of any given GP individual.

Larger individuals would have correspondingly larger numbers of expressed ERCs, which may increase the likelihood that ERCs might negatively affect the fitness of an individual. Figure 10.2 supports the notion that for the fraction of larger individuals that were affected by this altered context, these individuals were selected against (i.e., as depicted in shifts toward smaller individuals). Smaller individuals are more likely to occur near the outset of a GP run, as shown in Figure 10.3.

The second-half results showed that ERCs exhibit a nonuniform distribution and that those distributions are also affected by altering the low-level contexts in which ERCs appear. Furthermore, the nonuniform distributions are not well correlated with what one would intuit in solving for $f(x) = (x + 1)^3$. (In particular, we would reasonably expect values -1, 1, and maybe 0.)

We start this part of the discussion by comparing ERCs with other node types. On one hand, in comparison with these other types, ERCs do not appear remarkably different. Figure 10.4 shows that ERCs constitute roughly half of all the terminals used in the best of trial individuals for either Unity or Unity No-Null. On the other hand, the gross differences in the Figure 10.4 spider plot supports the earlier observation that Unity No-Null individuals are, on the whole, smaller than their Unity counterparts.

The smaller differences suggest that altering contexts also changes the way in which ERCs are maintained in GP individuals. Not only are the Unity No-Null individuals smaller, but the constituent node types are apportioned differently from their counterparts in Unity. In particular, the ratio of ERCs to `X` in Unity is 1:1; the corresponding ratio in Unity No-Null is 3:2. Figure 10.4 indirectly suggests that the underlying ERC PDFs would also differ.

We do offer a caveat in interpreting the ERC distributions between Unity and Unity No-Null. We noted earlier that at population initialization, the No-Null kernel replaces the second `X` in `( - X X)` with `r`. Although we felt that this replacement was minimally invasive, the replacement not transparent and increased the ratio of ERCs to `X` on the order of 5–6%. In Figure 10.4, we show comparative differences on that order. We emphasize, however,

that the replacement strategy used in the No-Null kernel occurred only at population initialization; the plots shown in Figure 10.4 (as well as Figure 10.5) were taken from results sometime after GP had run for several generations. We do not have, as of this writing, a full explanation of why these differences at population initialization were maintained, if in fact "maintenance" is what actually happened.[6]

Indeed, the term *maintenance* does not easily convey the GP dynamics that transform a uniformly distributed set of ERCs to the results shown in Figure 10.5. To appreciate the amount of transformation that occurs, one needs only to examine the trial data. As we mentioned earlier, each point represents the frequency (number of times) that *one* ERC value occurs in the best-of-trial population for a particular GP run. Typical values range from 1,000 to 10,000 counts. Those figures by themselves do not seem remarkable, unless one takes into account *that each of those figures arose from just one instance of that ERC value* at start. Amplifications for single ERC values usually meant amplification for ERCs as a whole in a given GP run, albeit not as dramatically; often just a fraction of the initial ERC values persisted until the best-of-trial generation. Nevertheless, the total number of ERCs in a best-of-trial population can easily be an order of magnitude more than at start.[7]

Not all ERC values were of equal worth. Figure 10.5b generally depicts what we would expect if ERCs were used as building blocks: enhanced distribution in the intervals of "useful" ERC values and diminished distribution in the intervals of "less-than-useful" ERC values. Noteworthy is that this pattern transcends approaches for solving for $f(x)$. In other words, this pattern occurred across Unity and Unity No-Null data sets, across the 600 populations represented by each data set, across the 30,000 individuals represented by those populations, and across the few thousand different approaches represented by those individuals. That the pattern exists is not in question, but what exactly determined ERC valuations of use?

It is not a given that ERCs are valued at all, since in Section 10.3 we established that GP can build solutions without ERCs. However from Figure 10.5b, we can surmise that at least part of ERC valuation comes from solving for $f(x)$. Note that in that figure, both Unity and Unity No-Null have similar, but not identical, distributions. We contend that these differences in distribution occurred largely because in Unity No-Null, we have essentially removed a means by which ERCs can hide in the introns of an individual. One consequence we mentioned in the discussion of the first-half results was that individuals were smaller in

[6]Note that population initialization does not account for the differences in the ratio of function types. At population initialization for both Unity and Unity No-Null, there were roughly equal amounts of each function type allocated at generation 0. In the best-of-trial individuals, that ratio changed substantially.

[7]Typically, about 4,500 to 5,000 ERCs are generated at population initialization using the GP parameters described in Section 10.2.2.

Unity No-Null than in Unity. Figure 10.5b supports another consequence: that more ERCs in Unity No-Null individuals were expressed than in Unity individuals. More ERCs expressed means that more ERCs contributed in solving for $f(x)$, which further implies that ERCs with greater worth in solving for $f(x)$ were selected for (and the corollary that ERCs with less worth were selected against). In terms of an ERC distribution, then, we would expect two key differences. First, we would expect Unity No-Null to have exhibited greater enhancements over those ERC values most used in solving for $f(x)$ than Unity would exhibit. Second, we would also expect Unity No-Null to have exhibited lower distributions over those ERC values least used in solving for $f(x)$ than Unity would exhibit. Figure 10.5b shows most of these two key differences in distributions between Unity No-Null and Unity—we consequently surmise that part of the ERC valuation came from solving for $f(x)$.

We would also contend, however, that ERC valuation was not well correlated with the task of solving for $f(x)$. The most useful ERC distributions in solving for $f(x)$ involve narrow peaks around ± 1, and maybe a peak at 0. In fact, Figure 10.5b shows a peak at 0 (for both Unity and Unity No-Null), but also shows peaks at roughly ± 0.75. It is not obvious why peaks exist at ± 0.75. The evidence suggests that these peaks were not statistical aberrations: both curves represent over fifty million ERCs. The evidence also suggests that these peaks were not artifacts from processing or smoothing: these peaks can also be found in the raw trial data and the numerical PDFs in Figure 10.5a. The data strongly suggests that the values around ± 0.75 have worth—the peaks are highly pronounced—but to what can that worth be ascribed?

In Section 10.2.1 and Appendix A.10.3, we mentioned that we might expect this type of behavior from ERCs. Namely, we might obtain a pattern in distribution that would transcend the approaches used to solve for $f(x)$ and that would also not be well correlated with the larger context for solving for that problem. Although there is not yet a formal analysis as of this writing, the derivation that led to this speculation in the first place may shed some insight on the direction such an analysis might take.

Our derivation, simply put, involved arbitrarily designating the GP Control case configuration (no ERCs) as host and ERCs as symbionts (parasites in this case).[8] The experiment configuration subsequently represented a symbiosis of host and parasites. The goal of the host was solve for $f(x)$. The goal of the parasites was to increase their progeny. If we were to extend this frame further, we could expect the following. The host attempts to shed its parasites, but does not have sufficient mechanisms by which to do so. Parasites attempt to infiltrate the host, but do so at the host's expense. An evolutionary stable strategy be-

[8]We defer to the original definition of symbiosis—*Zusammenleben ungleichnamiger Organismen* [de Bary 1879], relationships that are constant and intimate between dissimilar species. Under Anton de Bary's definition, symbiosis subsumes all systems of dissimilar species that live in this type of relationship, whether that system is characterized as mutualistic, commensal, or parasitic.

tween host and parasites is represented by the scenario whereby the host manages to fill its own niche while carrying as many parasites as is possible to survive in that niche.

Given this frame, it may well be that the peaks around ± 0.75 occurred not because those were the values most needed to solve for $f(x)$, but because those were the values that enabled the most numbers of ERCs to persist *in spite of* GP solving for $f(x)$. We leave exploring this conjecture to further work.

10.5 Implications for Building Blocks

Are ERCs building blocks?

In the intuitive sense of the term *building blocks* (i.e., simple components out of which more complex things can be made), our response would be a qualified yes, ERCs are building blocks. We have observed that GP assembled individuals that solved for $f(x)$ by using multiple instances of particular ERC values within an individual. This chapter has further shown that multiple instances of particular ERC values are distributed throughout a population.

We would not contend, however, that ERCs correspond to the view of building blocks as *context-free* structural schemata—the evidence did not totally support that. Context mattered. We showed that altering contexts changed the ability of GP to solve for $f(x)$ and modified the distribution of ERCs within a population. ERCs manifested themselves in different contexts throughout the course of a GP run—say, as "contributing value" (when creating numerical constants required by an individual) or as "don't care" (when appearing in an intron). At best, we would say that ERCs are structural schemata, which are sometimes used to build a solution.

Even the notion that ERCs are structural schemata may be open to debate. The prevailing view of structural schemata as genotype is questionable, as far as ERCs are concerned. The evidence in this chapter suggests that ERCs may be simultaneously both genotype (i.e., heritable information) and phenotype (i.e., behavior weakly coupled to solving for $f(x)$). It has not escaped our attention that our treatment of ERCs in this chapter does not preclude extension to other single-node subtree types (e.g., other terminals) or even encapsulated code (such as ADFs).

For that reason, we would contend that the assumption of a GP parse tree as a biological analog for real-world DNA may be tenuous. Goldberg and O'Reilly's comment in [Goldberg and O'Reilly 1998] was particularly insightful. The question, they claimed, is not so much what is a building block, but what is a gene. In biology, a gene maps to a particular physical structure. Heritable information and physical DNA structure are bijective quantities: a polypeptide generally maps to a particular gene, which is equivalent to saying that a polypep-

tide maps to a particular structure in DNA. As the results suggest, subtrees do not behave in this manner because of content and context dependency. In particular, we have demonstrated that an ERC can have germane information or can mean nothing at all depending on the particular location, content, context, and approach being employed in a given run. The mathematical structure that underlies a parse tree is fundamentally different from that of its biological counterpart—structure and heritable information in GP are not bijective. In the history of genetics, a gene was the heritable unit of information of which little was known. It just so happened that a biological gene maps bijectively to a DNA structure. In GP this is not the case. Consequently, to treat structures, whether single-node or multiple-node subtrees, as a biological gene is at best tenuous, and at worst, fallacious. We would say that usual metaphors of genotype and phenotype break down at this level of description. We subsequently argue that we would need to revisit the metaphors of genotype and phenotype.

We conjecture that unlike biology, building blocks in GP are ephemeral, just as the informational worth of an ERC changes over the course of a GP run. Building blocks are ephemeral because the projection of a parse tree into the space of worthwhile information would result in blocks that alternately appear and disappear.[9] Unlike the other implied metaphor of real building blocks, GP building blocks apparently phase in and out of worth. It isn't children's blocks that GP "plays" with—that scenario implies the existence of simple components to be available for use during the entire playing time. It's more like a cosmic game of Tetris, where one tries to solve a two-dimensional problem with N-dimensional blocks—so that now a block can seemly disappear at one moment and maybe reappear at another moment at another location with another shape.

10.6 Conclusions

This chapter has described the analysis of single-node subtrees, ephemeral random constants (ERCs), for a simple symbolic regression problem, which has as its target solution $f(x) = (x + 1)^3$. We have shown that in the intuitive sense of the term *building blocks* (i.e., simple components out of which more complex things can be made), our response would be a qualified yes, such single-node subtrees are building blocks.

We have demonstrated that we can manipulate the efficacy of GP to solve for $f(x)$ by adjusting the *contents* within the bounds of the allocated type corresponding to these single-node subtrees (as opposed to changing the structural complexity of an ERC or by increasing combinatorial search space). By so doing, we have demonstrated that these single-node

[9] This work supports O'Reilly and Oppacher's (1995) speculation on the idea of building blocks disappearing and reappearing, even though their discussion centered on building block disruption.

building blocks are significant to understanding the dynamics of GP for more than just their structural (single-node) aspect. Their intrinsic content mattered to the degree that different content resulted in different individual program sizes and depths.

We have also demonstrated that we can manipulate the efficacy of GP to solve for $f(x)$ by adjusting the *contexts* surrounding these single-node subtrees. By altering ERC contexts, we have shown that ERCs exhibit a pattern of distribution that transcends the many approaches that GP can use to solve for $f(x)$. We have also shown that this ERC distribution was not well correlated with the larger context of solving for $f(x)$ and have suggested an alternative frame based on a metaphor of symbiosis to account for this behavior. Using this frame, we have indicated that ERCs may be maintained in GP individuals not because they are useful in solving for $f(x)$, but because GP individuals allow for their existence. In other words, we have suggested that ERCs may exhibit dynamics that are only somewhat related to the selection pressure indicated by solving for $f(x)$.

Finally, we have discussed various implications of the results from this study in understanding what is a building block. As we have indicated, a building block in GP appears to be quite different from what we would expect to find in their biological counterpart. We contended that the assumption of a GP parse tree as a biological analog for real-world DNA may be tenuous. We subsequently argued that we would need to revisit the metaphors of genotype and phenotype with respect to their current usage in defining a GP building block.

For more information (other papers and code), please see our research group's site at http://www.sprl.umich.edu/acers.

Acknowledgments

The authors thank the editors L. Spector, W. Langdon, U.-M. O'Reilly, and P. Angeline for their kind invitation and support; W. Langdon and U.-M. O'Reilly, for their reviews, D. Ampy and S. Chang, for analysis tool support; D. Zongker and W. Punch for lilgp, S. Luke and P. Andersen for their patches to lilgp; M. Matsumoto and T. Nishimura for `mt19937.c`, their C implementation of the Mersenne Twister; and the original Challenges team of S. Ross, J. McClain, D. Ampy, and M. Holczer for doing early unpublished work on the empirical case study. We thank R. Riolo for his critique on an early draft of this chapter, A. Armstrong for allowing us to give a seminar on this chapter, as well as the following who gave additional reviews: S. Chaudhary, O. Chaudhri, G. Eickhoff, J. Khoo, H. Li, P. Litvak, M. Ratanasavetavadhana, and S. Yalcin. This chapter has benefited from our informal conversations with P. Angeline, W. Banzhaf, T. Bersano-Begey, D. Fogel, J. Foster, D. Goldberg, T. Haynes, W. Langdon, U.-M. O'Reilly, J. Rice, R. Poli, and J. Rosca. This research was partially supported through grants from U-M CoE, SPRL, NSF, OVPR, and UROP. We thank J. Vesecky, S. Gregerman, T. Killeen, and M. Combi for their continued support. The first author acknowledges I. Kristo and S. Daida. In memory of T. Daida, A. Bertram, F. Polito, and K. Daida.

Appendix A.10.1 Approaches to Solving $f(x)$

Although identifying the $f(x) = (x + 1)^3$ from 50 equidistant points distributed on the interval [-1,0) might seem straightforward, the means by which GP could obtain a solution—either perfect or approximate—are not. Some

solutions are concise, but that is more the exception than the rule. The ability to analyze the mappings between a solution and parse tree structure is not trivial, even given our problem. One can begin to appreciate the difficulty of this task by realizing that one approach (an approximate solution valid only on the given interval) consists of building a rational polynomial of order 50.

Of perfect solutions, there exist several approaches. A few of these approaches can be categorized as $(x + 1)^3$, $(1 + 2x + x^2)(x + 1)$, and $(1 + 3x + 3x^2 + x^3)$. (See Table 10.1.) For the purposes of this paper, we call these categories *transitive equivalence classes*. We define an equivalence class as a mathematical abstraction of structures that evaluates individuals to a particular expression. An equivalence class is not assumed to have the properties of closure, transitivity, associativity, and is not assumed to be distributive. However, in our case, we are not concerned about the ordering of factors (e.g., $(1 + 2x + x^2)(x + 1) = (x + 1)(1 + 2x + x^2)$ and $(1 + 3x + 3x^2 + x^3) = (x \times 3 + 1 + x^3 + 3x^2)$), so we use the term transitive equivalence class. For the remainder of this appendix, then, we assume that transitive equivalence and equivalence to mean the same, even though, technically speaking, they are somewhat different.

To illustrate the difficulty of exhaustively enumerating all GP approaches to solving for $f(x)$, we describe the approaches that can be taken with just those shown in Table 10.1. Note that each approach subsumes three different primary components: I, II, and III. Out of these primary components, come 56 total component variations, which includes both the primary components and close approximates. A close approximate is defined as a component that has parameter values that are within a specified tolerance of those in the corresponding primary component. Note, too, that we distinguish between implicit and explicit parameters. Terms like $1x$ and x are subsequently different, if only because the explicit parameter "1" maps to particular structures and the implicit "1" usually refers to an absence of associated structure. Note that Table 10.1 does not include the approaches that are purely approximate, which include solutions of more than three zeros and rational polynomials.

For the sake of completeness, there are nine other equivalence classes, in addition to the three shown in Table 1, that are based on addition and multiplication. These classes include the following: $(x + 1)(1 + x + x + x^2)$, $(1 + x + x + x + x^2 + x^2 + x^2 + x^3)$, $(1 + 2x + x + x^2 + x^2 + x^2 + x^3)$, $(1 + 3x + x^2 + x^2 + x^2 + x^3)$, $(1 + x + x + x + 2x^2 + x^2 + x^3)$, $(1 + x + x + x + 3x^2 + x^3)$, $(1 + 2x + x + 2x^2 + x^2 + x^3)$, $(1 + 2x + x + 3x^2 + x^3)$, $(1 + 3x + 2x^2 + x^2 + x^3)$. There are also further equivalence classes based on other permutations of subtraction, division, multiplication, and addition.

An equivalence class in GP is rarely bijective with an associated parse tree structure. As demonstrated in our particular problem, multiple structures can be mapped to a single equivalence class. For example, the equivalence class that corresponds to the value "1" subsumes structures like `(÷ X X)`, `(÷ X 0)`, `(÷ r 0)`. It is also true that multiple equivalence classes can be mapped to a single structure. For example, the structure `(÷ X X)` belongs as a partial solution to equivalence classes "3," "2," "1," and even "0." We defer to [Koza 1992] for further discussion of the building of numerical constants with GP.

As mentioned previously, each equivalence class presumes a different approach and different blocks for solution generation. That roots form a basis for solutions for many of these approaches is not surprising: root finding is a well established numerical technique. Finding roots is also a rapid way to come close to a target function, since a factor intersects with at least part of the target solution. Consequently, finding roots is one way to establish part of the "shape" of a solution.

Note that this analysis has been extended just to perfect solutions and their close approximates. It does not include any other types of approximations with, say, those that are purely approximate, which include solutions of more than three roots and rational polynomials. Purely approximate polynomials can be used to fit the fitness cases of 50 equidistant points on the interval [-1, 0) within a given tolerance bound, but not necessarily to fit any other points taken from $f(x)$. A common rational polynomial approach has been to place poles in the intervals outside of [-1,0), although some approaches feature poles inside the interval [-1, 0) and between points from the fitness case. A conservative estimate on the number of equivalence classes with approximate approaches, including rational polynomials, is on the order of a few thousand.

Appendix A.10.2 Known ERC Strategies

From a listing of equivalence classes, one could infer that there are only a few low-level strategies that are required to solve the problem of ERCs: incorporation, elimination, and absorption. An ERC can be *incorporated* into an individual by serving as a component parameter. For example, an ERC with value 0.99 could find use wherever a unity value is required. An ERC can be *eliminated* from an individual via crossover. An ERC can also be eliminated from further consideration (i.e., a population) if individuals bearing this ERC neither reproduce nor replicate

Table 10.1
Detail of perfect and close approximate approaches using addition and multiplication.

	Class		
	$(x+1)^3$	$(x+1)(x^2+2x+1)$	(x^3+3x^2+3x+1)
Primary Component	**I**	**I, II**	**III**
Variants	$\mathbf{I}_{-\rightarrow}^{1}$, $\mathbf{I}^{1,1}$, $\mathbf{I}_{\gamma_1}^{,1}$, $\mathbf{I}_{-\rightarrow\gamma_1}$, $\mathbf{I}^{1,\gamma_1}$, $\mathbf{I}_{\gamma_1}^{,\gamma_1}$, $\mathbf{I}_{\gamma_1}^{,\gamma_2}$	$\mathbf{I}_{-\rightarrow}^{1}$, $\mathbf{I}^{1,1}$, $\mathbf{I}_{\gamma_1}^{,1}$, $\mathbf{I}_{-\rightarrow\gamma_1}$, $\mathbf{I}^{1,\gamma_1}$, $\mathbf{I}_{\gamma_1}^{,\gamma_1}$, $\mathbf{I}_{\gamma_1}^{,\gamma_2}$	$\mathbf{III}_{-\rightarrow}^{a,b,1}$, $\mathbf{III}^{1,a,b,1}$, $\mathbf{III}_{\gamma_1}^{,a,b,1}$, $\mathbf{III}_{-\rightarrow}^{a,b,\gamma_1}$, $\mathbf{III}^{1,a,b,\gamma_1}$, $\mathbf{III}_{\gamma_1}^{,a,b,\gamma_1}$, $\mathbf{III}_{\gamma_1}^{,a,b,\gamma_2}$, where $\mathbf{a} \in \{3, \eta_1\}$ and $\mathbf{b} \in \{3, \eta_1\}$
		$\mathbf{II}_{-\rightarrow}^{a,1}$, $\mathbf{II}^{1,a,1}$, $\mathbf{II}_{\gamma_1}^{,a,1}$, $\mathbf{II}_{-\rightarrow}^{a,\gamma_1}$, $\mathbf{II}^{1,a,\gamma_1}$, $\mathbf{II}_{\gamma_1}^{,a,\gamma_1}$, $\mathbf{II}_{\gamma_1}^{,a,\gamma_2}$, where $\mathbf{a} \in \{2, \lambda\}$	$\mathbf{III}_{-\rightarrow}^{,\eta_1,\eta_2,1}$, $\mathbf{III}^{1,\eta_1,\eta_2,1}$, $\mathbf{III}_{\gamma_1}^{,\eta_1,\eta_2,1}$, $\mathbf{III}_{-\rightarrow}^{,\eta_1,\eta_2,\gamma_1}$, $\mathbf{III}^{1,\eta_1,\eta_2,\gamma_1}$, $\mathbf{III}_{\gamma_1}^{,\eta_1,\eta_2,\gamma_1}$, $\mathbf{III}_{\gamma_1}^{,\eta_1,\eta_2,\gamma_2}$
Number Variants	**I(7)**	**I(7), II(14)**	**III (35)**
Root Factors	**3**	**1**	**0**
Parameter Values	**1**	**1,2**	**1,3**

for the next generation. An ERC can be *absorbed* by being part of a structure that is, for all practical purposes, not expressed. For example, an ERC could be multiplied by the null structure `( - X X)` so that the overall subtree evaluates to null. In GP jargon, structures that do not directly affect a solution's fitness are called introns (after [Angeline 1994]). (Both theoretical and empirical evidences support the notion that introns emerge because genetic operators can result in decreased fitness [Nordin and Banzhaf 1995; Rosca and Ballard 1995; Nordin et al. 1996; Soule et al. 1996; Banzhaf et al. 1998]. Introns tend to grow exponentially in numbers. Introns may have differing effects before and after exponential growth of introns begins [McPhee and Miller 1995; Banzhaf et al. 1998]. Opinion on whether introns benefit or hinder GP is moot, since supportive research can be found on either side of the subject, (e.g., [Andre and Teller 1996] versus [Nordin et al. 1996]).)

There are 12 different ways to construct either an approximate zero or a null structure, for trees up to depth 2 and for $r \cong 0$ (where $r \in \mathbf{R}$) using a tolerance of 0.01. Note that the most likely structure `( - X X)` to appear out of these ways also represents the only structure that evaluates to a perfect null: probability P(structure|depth) = 6.25% for `( - X X)`, in comparison to the next highest probability structure, an approximate zero, which has a P(structure|depth) = 1.36%. The remaining approximate zero structures have P(structure|depth) no greater than 0.5%.

Note that by multiplying a subexpression with a perfect null, any number of ERCs can be absorbed. In contrast, an approximate null has only a limited ability to absorb ERC values through multiplication. By dividing a subexpression with a perfect null, again any number of ERCs can be absorbed, but with the added benefit that protected division returns unity. (The value "1" can subsequently be used as a component parameter in variants of I, II, or III.) Dividing a subexpression with an approximate null would likely result in anything but unity.

Appendix A.10.3 Alternative Frame for Analyzing GP and ERCs

Conventional wisdom concerning ERCs would suggest that ERCs are but another software component in GP from which to build solutions. Upon second look, though, we found GP using ERCs as reminiscent of a class of hybrid systems that we had been studying for several years [Daida et al. 1995; Daida et al. 1996] and have developed several systems around the concept. Although we used the concept for synthesis, it occurred to us that we could use the associated theoretical concepts developed in [Daida et al. 1995] as an alternative frame for this chapter's analysis.

Our theoretical framework has used biological symbiosis—relationships that are constant and intimate between dissimilar species—as a metaphor. In [Daida et al. 1995], we define a symbiotic system as follows:

Let system **S** represent a set of adaptive systems

$$S = \bigcup_{\forall i \in \Sigma} \left\{ S^i : S^i = \left(A^i, \Omega^i, \tau^i, B^i, \Omega_B^i, \tau_B^i, I^i \right) \right\},$$

where Σ is an index set corresponding to elements in **S**, **A** is the set of attainable structure and is the domain of action for the adaptive plan, Ω is the set of operators for modifying structures **A**, τ is the adaptive plan that determines what operator is to be applied, **B** is the set of attainable artifacts generated by the adaptive system. $\Omega_{\mathbf{B}}$ is the set of operators for modifying artifacts, $\tau_{\mathbf{B}}$ is the adaptive plan that determines what operator is to be applied to **B**, and *I* is the set of possible inputs to the system from the environment. System **S** is symbiotic if and only if there exists an instance where

$$B^i \supset I^j, \exists_i : i \neq j, \forall_{i,j \in \Sigma} : i \neq j, \text{ and } \bigcup_{\forall k \in \Sigma} B^k \neq \varnothing.$$

A symbiotic system **S** is considered minimal if and only if only one element $\mathbf{S}^i$ describes an adaptive system.

We use the concept of a minimal symbiotic system to create our alternative frame of analysis. To do so, we arbitrarily define a minimal symbiotic system **S** that consists of two component systems $\mathbf{S}^1$ and $\mathbf{S}^2$. In our case we designate $\mathbf{S}^1$ as the adaptive system and $\mathbf{S}^2$ as the non-adaptive one. We let $\mathbf{S}^1$ be a GP system (adaptive) with the terminal set = {x, Γ}, where Γ is a set of indexed terminal placeholders. We further let $\mathbf{S}^2$ be an ERC generator (non-adaptive) that outputs {ρ}, where ρ is an indexed set of random number values that are uniformly distributed in the interval $[-a_{\mathbf{R}}, a_{\mathbf{R}}]$. Under this definition, we share the set of indices that point to Γ and ρ such that indexes in Γ serve as input to ρ and indexes in ρ serve as input to Γ. We let indexes in Γ serve as pointers, and indexes in ρ serve as addresses. In a sense, the shared indices serve as simple tags, numerical labels that formed the interface between the two systems.

Framing the GP system (and ERC generator) in this way has been helpful because it is similar to one we studied in [Daida et al. 1995]. That system **S** also had two component systems $\mathbf{S}^1$ and $\mathbf{S}^2$, where $\mathbf{S}^1$ was a genetic algorithm and $\mathbf{S}^2$ an arbitrary, but fixed heuristic. Both component systems interacted via numerical tags. For instance, for $\mathbf{S}^2$, the numerical tags served as "locks," while for $\mathbf{S}^1$, numerical tags served as "keys." Locks and keys were independently generated by either system. Behind every lock was a vector that was generated by the fixed heuristic. Behind every key was a function that used elements from an indexed vector (like one generated by the fixed heuristic). Although matching lock and key sets could be rapidly determined, the task of discovering "useful" content was left to $\mathbf{S}^1$. We showed in that experiment that the selection of locks and keys was not deterministic, was subject to positive feedback, and was opportunistic. The pattern of which heuristics were chosen also seemed probabilistic—$\mathbf{S}^1$ would build a solution around a vector by choosing a vector first, rather finding the best attainable solution and then finding the heuristic that would make further refinements. Given that type of behavior, we were able to demonstrate a few ways to manipulate the search dynamics of $\mathbf{S}^1$ through tags.

In the GP system that we used, keys would be the kernel's pointers to ERCs; locks would be the hash table associated with ERC values. ERC values are generated once at population initialization. Unlike the system described in [Daida et al. 1995], each GP individual that uses ERCs uses not one, but a set of keys. Our hypothesis, however, was that we would find a similar type of behavior.

Bibliography

Altenberg, L. (1994). The Evolution of Evolvability in Genetic Programming. In K. E. Kinnear, Jr. (Ed.), *Advances in Genetic Programming* (pp. 47–74). Cambridge: The MIT Press.

Andre, D. and A. Teller (1996). A Study in Program Response and the Negative Effects of Introns in Genetic Programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference: July 28–31, 1996, Stanford University* (pp. 12–20). Cambridge: The MIT Press.

Angeline, P. (1994). Genetic Programming and Emergent Intelligence. In K.E. Kinnear, Jr. (Ed.), *Advances in Genetic Programming* (pp. 75–97). Cambridge: The MIT Press.

Angeline, P. J. (1997). Subtree Crossover: Building Block Engine or Macromutation? In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University* (pp. 9–17). San Francisco: Morgan Kaufmann Publishers, Inc.

Bäck, T. and D. B. Fogel (1997). Glossary. In T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.), *Handbook of Evolutionary Computation* (pp. Glos:1–Glos:10). Bristol: Institute of Physics Publishing.

Banzhaf, W., P. Nordin, R. E. Keller, and F. D. Francone (1998). *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. San Francisco, Morgan Kaufmann Publishers, Inc.

Daida, J. M., C. S. Grasso, S. A. Stanhope, and S. J. Ross (1996). Symbionticism and Complex Adaptive Systems I: Implications of Having Symbiosis Occur in Nature. In L. J. Fogel, P. J. Angeline and T. Bäck (Eds.), *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming* (pp. 177–86). Cambridge: The MIT Press.

Daida, J. M., S. J. Ross, and B. C. Hannan (1995). Biological Symbiosis as a Metaphor for Computational Hybridization. In L. J. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 328–35). San Francisco: Morgan Kaufmann Publishers, Inc.

Daida, J. M., S. J. Ross, J. J. McClain, D. S. Ampy, and M. Holczer (1997). Challenges with Verification, Repeatability, and Meaningful Comparisons in Genetic Programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University* (pp. 64–9). San Francisco: Morgan Kaufmann Publishers, Inc.

de Bary, A. (1879). *Die Erscheinung der Symbiose. Vortrag, Gehalten auf der Versammlung Deutscher Naturforscher und Aerzte zu Cassel.* Strassburg: R.J. Trübner.

Fogel, D. B. (1992). A Brief History of Simulated Evolution. In *The First Annual Conference on Evolutionary Programming* (pp. 1–16). San Diego: Evolutionary Programming Society.

Fuchs, M. (1998). Crossover versus Mutation: An Empirical and Theoretical Case Study. In J. R. Koza, W. Banzhaf, K. Chellapilla, et al. (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22–25, 1998, University of Wisconsin, Madison* (pp. 78–85). San Francisco: Morgan Kaufmann Publishers, Inc.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Addison-Wesley Publishing Company, Inc.

Goldberg, D. E. and U.-M. O'Reilly (1998). Where Does the Good Stuff Go, and Why? In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (Eds.), *Proceedings of the First European Conference on Genetic Programming, Paris, France*. Berlin: Springer-Verlag.

Haynes, T. (1997). Phenotypical Building Blocks for Genetic Programming. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 26–33). San Francisco: Morgan Kauffmann Publishers.

Hellekalek, P. (1997). A Note on Pseudorandom Number Generators. *Simulation Practice and Theory* 5(6): 6–8.

Hellekalek, P. (1998). Good Random Number Generators Are (Not So) Easy to Find. *Mathematics and Computers in Simulation* 46(5–6): 487–507.

Holland, J. H. (1973). Genetic Algorithms and the Optimal Allocation of Trials. *SIAM Journal on Computing* 2(2): 88–105.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, University of Michigan Press.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* Cambridge, The MIT Press.

Iba, H. and H. de Garis (1996). Extending Genetic Programming with Recombinative Guidance. In P. J. Angeline and K.E. Kinnear, Jr. (Eds.), *Advances in Genetic Programming* (pp. 69–88). Cambridge: The MIT Press.

Ito, T., H. Iba, and S. Sato (1998). Depth-Dependent Crossover for Genetic Programming. In *The 1998 IEEE International Conference on Evolutionary Computation Proceedings: IEEE World Congress on Computational Intelligence* (pp. 775–80). Piscataway: IEEE Press.

Koza, J. R. (1989). Hierarchical Genetic Algorithms Operating on Populations of Computer Programs. In N. S. Sridharan (Ed.), *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 768–74). San Francisco: Morgan Kaufmann.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, The MIT Press.

Langdon, W. B. and R. Poli (1998). Why Ants Are Hard. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22–25, 1998, University of Wisconsin, Madison* (pp. 193–201). San Francisco: Morgan Kaufmann Publishers, Inc.

Luke, S. and L. Spector (1998). A Revised Comparison of Crossover and Mutation in Genetic Programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, et al (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22–25, 1998, University of Wisconsin, Madison* (pp. 208–13). San Francisco: Morgan Kaufmann Publishers, Inc.

Macready, W. G. and D. H. Wolpert (1998). Bandit Problems and the Exploration/Exploitation Tradeoff. *IEEE Transactions on Evolutionary Computation* 2(2): 2–22.

Matsumoto, M. and T. Nishimura (1997). *mt19937.c*. Keio, Department of Mathematics, Keio University. http://www.math.keio.ac.jp/~matumoto/emt.html.

Matsumoto, M. and T. Nishimura (1998). Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudorandom Number Generator. *ACM Transactions on Modeling and Computer Simulation* 8(1): 3–30.

McPhee, N. F. and J. D. Miller (1995). Accurate Replication in Genetic Programming. In L. J. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 303–309). San Francisco: Morgan Kaufmann Publishers, Inc.

Mulloy, B. S., R. L. Riolo, and R. S. Savit (1996). Dynamics of Genetic Programming and Chaotic Time Series Prediction. In J. R. Koza, D. E. Goldberg, D. B. Fogel and R. L. Riolo (Eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference: July 28–31, 1996, Stanford University* (pp. 166–74). Cambridge: The MIT Press.

Nordin, P. and W. Banzhaf (1995). Complexity Compression and Evolution. In L. J. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 310–17). San Francisco: Morgan Kaufmann Publishers, Inc.

Nordin, P., F. Francone, et al. (1996). Explicitly Defined Introns and Destructive Crossover in Genetic Programming. In P. J. Angeline and K.E. Kinnear, Jr. (Eds.), *Advances in Genetic Programming* (pp. 111–34). Cambridge: The MIT Press.

O'Reilly, U.-M. and F. Oppacher (1995). The Troubling Aspects of a Building Block Hypothesis for Genetic Programming. In L. D. Whitley and M. D. Vose (Eds.), *Foundations of Genetic Algorithms 3* (pp. 73–88). San Francisco: Morgan Kaufmann Publishers, Inc.

Poli, R. and W. B. Langdon (1997). An Experimental Analysis of Schema Creation, Propagation and Disruption in Genetic Programming. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 18–25). San Francisco: Morgan Kauffmann Publishers, Inc.

Poli, R. and W. B. Langdon (1997). A New Schema Theory for Genetic Programming with One-Point Crossover and Point Mutation. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University* (pp. 279–85). San Francisco: Morgan Kaufmann Publishers, Inc.

Poli, R., W. B. Langdon, and U.-M. O'Reilly (1998). Analysis of Schema Variance and Short Term Extinction Likelihoods. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22–25, 1998, University of Wisconsin, Madison* (pp. 284–92). San Francisco: Morgan Kaufmann Publishers, Inc.

Punch, W., D. Zongker, and E. Goodman (1996). The Royal Tree Problem, A Benchmark for Single and Multiple Population Genetic Programming. In P. J. Angeline and K.E. Kinnear, Jr. (Eds.), *Advances in Genetic Programming* (pp. 299–316). Cambridge: The MIT Press.

Rosca, J. P. (1997). Analysis of Complexity Drift in Genetic Programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University* (pp. 286–94). San Francisco: Morgan Kaufmann Publishers, Inc.

Rosca, J. P. and D. H. Ballard (1995). Causality in Genetic Programming. In L. J. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 256–63). San Francisco: Morgan Kaufmann Publishers, Inc.

Rosca, J. P. and D. H. Ballard (1996). Discovery of Subroutines in Genetic Programming. In P. J. Angeline and K.E. Kinnear, Jr. (Eds.), *Advances in Genetic Programming* (pp. 177–201). Cambridge: The MIT Press.

Soule, T., J. A. Foster, and J. Dickinson (1996). Code Growth in Genetic Programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference: July 28–31, 1996, Stanford University* (pp. 215–23). Cambridge: The MIT Press.

Soule, T., J. A. Foster, and J. Dickinson (1996). Using Genetic Programming to Approximate Maximum Clique. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference: July 28–31, 1996, Stanford University* (pp. 400–405). Cambridge: The MIT Press.

Tackett, W. A. (1993). Genetic Programming for Feature Discovery and Image Discrimination. In S. F. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 303–309). San Mateo: Morgan Kaufmann Publishers, Inc.

Tackett, W. A. (1994). *Recombination, Selection and the Genetic Construction of Computer Programs*. Ph.D. Thesis, Electrical Engineering. Los Angeles, University of Southern California.

Whigham, P. (1995). A Schema Theorem for Context-Free Grammars. In *The 1995 IEEE Conference on Evolutionary Computation* (pp. 178–81). Piscataway: IEEE Press.

Zongker, D. and W. Punch (1995). *lilgp*. Lansing, Michigan State University, Genetic Algorithms Research and Applications Group. http://garage.cps.msu.edu/software/software-index.html.