Justinian P. Rosca and Dana H. Ballard

In this chapter we present a novel way of addressing the issue of variable complexity of evolved solutions and a revised interpretation of how Genetic Programming (GP) constructs solutions, based on the rooted-tree schema concept.

A rooted-tree schema is a simple relation on the space of tree-shaped structures which provides a quantifiable partitioning of the search space. Formal manipulation of rooted-tree schemata allows: (1) The role of the size in the selection and survival of evolved expressions to be made explicit; (2) The interrelationship between parsimony penalty, size, and fitness of evolved expressions to be clarified and better understood; (3) The introduction of alternative approaches to evolving parsimonious solutions by preventing rooted-tree schema from bloating.

The rooted-tree schema concept provides a top-down perspective of how program expressions are evolved, contrary to the common belief that small pieces of code, or building blocks, are gradually assembled to create solutions. Analysis shows that GP, while it improves solutions, combines both bottom-up and top-down refinement strategies.

11.1 Introduction

Complexity (or the size or length) of evolved structures is a non-issue in most of the recent evolutionary computation (EC) literature. EC techniques such as genetic algorithms (GAs) [Holland, 1992], evolutionary programming (EP) [Fogel et al., 1966; Fogel, 1995], and evolution strategies (ES) [Bäck et al., 1991] use mostly fixed length encodings for the structures to be evolved. This design decision seriously limits their applicability solely to the domain of parametric problems. Many applications could benefit enormously from simulated evolution that is open-ended with respect to the complexity of evolved solutions, i.e. when no particular structure is assumed *a priori*. This would be particularly the case with complex design or control problems where the structure of a satisfactory solution is unknown in advance.

Genetic programming (GP) uses open-ended complexity representations that have flexible semantics [Koza, 1992]. GP evolves a population of programs in some problem dependent language in the form of tree expressions of variable length (complexity) and shape that encode solutions to the problem. A program can be used to generate practically any computation such as classification, regression, prediction, or control, side effect, or create a secondary structure representing the learned model. For example, the program can construct the architecture and the corresponding number of parameters of a model such as a belief network or a neural network. Thus, evolved programs are very general non-parametric encodings, in the sense that the number of parameters can be very large and their role is very flexible.

The behavior of GP is extremely hard to characterize formally. Yet expanding the range of applications to new problems and domains requires a qualitative understanding of its behavior. This goal can be furthered by a careful analysis of the characteristics and limita-

tions of the search process. In this contribution we focus on the specific characteristics and limitations of GP with variable complexity representations.

GP searches the space of programs. At every search step, the GP engine ranks the population of program expressions according to a problem dependent measure, the fitness function. Then it applies fitness proportionate selection and random variation of expressions in the population in order to generate a next step population. The complexity of evolved expressions can drastically vary over the span of the search.

One serious problem of standard GP is that evolved expressions appear to drift towards larger forms that take longer to evaluate on average. This threatens to limit the performance of the GP engine because prespecified threshold parameters such as the total size or depth of manipulated expressions are met [Sims, 1991; Tackett, 1994; Angeline, 1994; Nordin et al., 1995; Rosca, 1996; Soule et al., 1996; Langdon and Poli, 1998; Langdon and Poli, 1997]. Performance near the edge of these parameters is not desirable. The simple-minded solution is to monitor when such non-desirable operating conditions are reached and to increase the values of the relevant parameters to avoid such conditions. The GP run could be continued if resources still allow. Another solution to control size growth is to include a parsimony penalty component into the fitness function in order to limit the growing size of expressions [Iba et al., 1994; Rosca and Ballard, 1994; Zhang and Muhlenbein, 1995]. However, the penalty component changes the fitness landscape. How heavily should the parsimony penalty be weighted or how should it be adapted in order to not affect the underlying optimization process?

This chapter presents a novel view on the role played by size growth in evolutionary computation. It discusses a particular property of GP representations that sheds light on the role of variable complexity of evolved structures. The property of interest is the *rooted-tree schema* relation on the space of tree-shaped structures which provides a quantifiable partitioning of the search space. The analysis answers questions such as: What role does variable complexity play in the selection and survival of structures? What is the influence of parsimony pressure on selection and survival of structures? What is the role played by the weighting factor of the complexity term? Are there alternative approaches to imposing parsimony pressure that do not result in a change of the fitness landscape? The paper provides theoretical answers to these questions, interpretation of these results and an experimental perspective. Answers to such questions are considered critical in the challenging attempt of understanding and controlling the behavior of evolutionary computation algorithms operating with variable complexity representations and on future work in EC.

The structure of the chapter is as follows. Section 11.2 overviews schema theory [Holland, 1992] and attempts to define the notion of GP schema. Section 11.3 defines the rooted-tree schema relation. It then develops a theoretical analysis of the role played by the variable complexity of evolved representations and discusses interpretations of the theory. Next, Section 11.4 analyzes the effect of parsimony penalty on selection and in the tree-schema growth formulae. Section 11.5 discusses two alternative ways for controlling schema growth. An alternative based on dynamic adaptation of mutation and crossover

probabilities will be experimentally examined in the following section. Section 11.6 discusses results of simulations of the standard GP algorithm with and without a parsimony penalty and presents an adaptive algorithm for controlling complexity. Based on the rooted-tree schema theory, Section 11.7 further discusses how the ideas of competing schemata and refinement of schemata suggest a top-down theory of solution acquisition. Section 11.8 reviews related work from the machine learning perspective, in general, and the EC perspective, in particular. Finally, section 11.9 summarizes this work.

11.2 Schema Theory

11.2.1 Schemata in genetic algorithms

Of all theoretical studies on the dynamics of a simple genetic algorithm, *schema theory* and the *building block hypothesis* have been the most controversial. In this section we review schema theory and its extension to GP, in order to contrast the new approach to be described later.

The concept of *schema* was introduced by Holland in order to characterize the informal notion of "building block." It was intuitively thought that GAs work by recombining building blocks. For linear binary representation of fixed length L, a schema is defined by a string of length L over the binary alphabet extended with a *don't care* symbol. A schema is interpreted as a template string whose 0 and 1 bits represent a fragment, or block of a chromosome.

The Schema Theorem gives an estimate of the change in the frequency of a schema in the population as a result of fitness-proportionate reproduction, crossover, and mutation ([Holland, 1992], see also [Goldberg, 1989]). We will restate it below.

Let $\mathcal{H}$ be a fixed schema of defining length $\delta(\mathcal{H})$ and $m(\mathcal{H}, t)$ be the number of copies of $\mathcal{H}$ in the population at time (generation) t. Let $f_{\mathcal{H}}$ be the average fitness of all strings in the population matching $\mathcal{H}$ and f be the average fitness of the population. Consider a simple GA using fitness proportionate selection for reproduction. Offspring are created through copying or single-point crossover, and additional variation through mutation. The probabilities for crossover and mutation are respectively p_c and p_m. Then, a lower bound on the number of copies of $\mathcal{H}$ in the next generation is given by [Goldberg, 1989]:

$$m(\mathcal{H}, t+1) \geq m(\mathcal{H}, t) \cdot \frac{f_{\mathcal{H}}}{f} \left(1 - p_c \frac{\delta(\mathcal{H})}{L-1}\right) (1 - p_m)^{o(\mathcal{H})} \qquad (11.1)$$

The coefficient of $m(\mathcal{H}, t)$ on the right hand side of relation (11.1) represents an approximation of the growth factor of schema $\mathcal{H}$. The constructive effect of variation operations was not considered. A super-unitary growth factor indicates that the next generation will contain more samples of the schema.

The theorem has been interpreted as showing that schemata with fitness values above population average, of low order and short length – all intuitive conditions for a super-unitary growth factor – will receive an exponentially increasing number of samples in the next generations. Such schemata are *building blocks*. Good individuals tend to be made up of good schemata, i.e. building blocks. The GA discovers and recombines such building blocks in parallel to create solutions. This is the essence of the Building Block Hypothesis as presented in [Goldberg, 1989]. Moreover, Holland argued that the search for an optimal string combines exploitation (preservation of schemata) and exploration (creation of new schemata) in close to an optimal proportion. The argument relied on the analogy between the allocation of samples to schemata in the GA with the allocation of effort in the Two-Armed-Bandit problem [Holland, 1992].

Schema theory has been criticized for not reflecting the processing done by a GA and not being really informative. One such critique is that GA allocates trials to schemata very differently from the optimal allocation given by the Two-Armed-Bandit solution. This was shown on contrived examples [Grefenstette and Baker, 1989; Mühlenbein, 1991]. Some of the discussions of schema theory caveats, such as that schema frequency variation is in disharmony with the Schema Theorem, are summarized in [Mitchell, 1996]. The problem with current interpretations is that they do not take into account what are independent schemata. Schema interdependence is due to inclusion relationships and epistasis. A formal notion of schema independence should take into account such effects. Independence can be defined relative to fitness contribution. Analysis and interpretations should therefore focus on the relevant entities, i.e. the independent competing schemata.

Another critique is that schemata do not necessarily capture relationships among meaningful properties that determine fitness [Altenberg, 1995]. Generalized schemata can be defined by partitioning the space of structures with many other relations. Such attempts have been presented in the GA literature [Vose and Liepins, 1991; Radcliffe, 1991]. Relations analogous to the schema theorem will hold for other representations as well [Radcliffe, 1992]. Indeed, schema theory explains the proliferation of substructures through selection. However, such properties can rather be used to refute a schema hypothesis. An argument for this remark is the intended use of Price's covariance and selection theorem [Price, 1970]. Price's theorem states that the variation in the frequency of a gene between the offspring and the parent population is proportional to the covariance between the frequency of the gene in an individual and the number of offspring of that individual over the parent population. Price outlined that his theorem helps in constructing hypotheses about selection, such as whether a certain behavioral feature is advantageous:

> Recognition of covariance is of no advantage for numerical calculation, but of much advantage for evolutionary reasoning and mathematical model building [Price, 1970, page 521].

Altenberg [Altenberg, 1995] brought attention on Price's theory and its general implications in GA theory. Although he dismissed the merit of current interpretations of the

schema theory in explaining GA performance, he pointed out that the Schema Theorem is a particular case of Price's covariance theorem.[1] Altenberg further generalized the view of Price's theorem by considering *measurements* other than the frequency of a gene (schema).

Both schema theory and Price's theorem explain the variation in the frequency of a schema (Holland) or a group of genes (Price) over successive generations, but they rely on different arguments. Holland concentrates on the effects of selection, reproduction, crossover and mutation, so that his derivation explicitly incorporates fitness due to its role in selection. Price's analysis relies, more generally, on the correlation between the number of offspring produced from parents and the frequency of a certain gene in the parent population. Also, both theorems can be used to construct hypothesis about the evolutionary process, rather than fully explain its dynamics.

In Section 11.3 we introduce a new relation on the space of programs that allows us to focus on the interaction between fitness and complexity in variable complexity representations. Our measurement is the frequency of individuals in the population satisfying the relation. Before that, we critically examine approaches to extend the schema notion to GP.

11.2.2 Schemata in genetic programming

GA schemata have been interpreted from two interchangeable perspectives. The first perspective focuses on what genome parts can remain unchanged after repeated crossover operations and how these parts are assembled through recombination. The building block hypothesis shares this view. The second perspective interprets a schema as the set of individuals sharing common constraints (the defined elements) on their structure. By creating schemata with more defined elements, i.e. of higher order, the GA focuses search on smaller regions of the space.

It is not at all obvious how the GA schemata interpretations can be transferred to the variable length structures manipulated by GP. What property of variable length representations would be useful to observe or analyze in order to explain changes in structure or fitness and predict the dynamics of GP?

Again one can take two different perspectives of a GP schemata. The first focuses on the structural variation through crossover and mutation while the second focuses on subsets of the search space. In GA schema theory the two views were just the two sides of the same coin. In GP each of the two interpretations leads to different definitions and suggests different insights. Next we will review two definitions that take the first perspective and then present a new approach, the rooted-tree schema theory, that takes the second perspective.

GP searches the space of trees constrained by a maximum size or depth of trees. The tree representation can be also viewed as a string made of function and terminal symbols. The GP string has a default parenthesized structure imposed by the original hierarchical

[1] To see how an approximation of the Schema Theorem can be derived from Price's theorem, take into account that in a GA the number of offspring containing a gene (and its correspondent, a schema) is correlated with the deviation of fitness from the average population fitness for fitness proportional selection.

shape. Although the GA schema concept cannot be directly applied to the parenthesized string, it is helpful to think of GA schema as sets of fixed-length bit strings that have a number of features (bits) in common and then extend the definition to GP by considering sets of programs that have features in common. The question that remains is what kind of features would be most informative. Koza's schema definition focuses on subtrees as features. Programs in a GP schema set have in common one or more specified subtrees:

> A schema in genetic programming is the set of all individual trees from the population that contain, as subtrees, one or more specified trees. A schema is a set of LISP S-expressions sharing common features [Koza, 1992, page 117].

While a GA schema implicitly specifies through bit positions how two low order schemata are to be combined, this GP schema definition does not. Subtrees defining a schema can appear anywhere within the structure of an individual that belongs to the schema, provided that the entire structure obeys a maximum size or depth constraint.

The definition above suggests the intuitive idea that subtrees may play the role of functional features and that good features may be functionally combined to create good representations [Rosca and Ballard, 1996a].

Another, more general, GP schema definition is provided in [O'Reilly and Oppacher, 1995]. The previous definition allows for trees to be combined only as subtrees in larger structures. This second definition makes explicit how schema components can be combined in larger structures in analogy to a GA schema by using wildcards.

> A GP-schema H is a set of pairs. Each pair is a unique S-expression tree or fragment (i.e. incomplete S-expression tree with some leaves as wildcards) and a corresponding integer that specifies how many instances of the S-expression tree or fragment comprise H. [[O'Reilly and Oppacher, 1995], page 77]

While the Koza definition considers only the type of component manipulated by the crossover operation, this definition allows for the incomplete specification of fragments of a tree "corresponding to what is left intact by repeated crossovers." The extension allows a schema such as one or more contiguous fragments of a tree, where each tree fragment may have wildcards at its root or on its leaves.[2] Additionally, a duplication factor is considered for each fragment. In contrast to a GA schema, there is no predetermined way to combine the fragments.

Both definitions support the intuition that subtrees may be building blocks. However, the problem with both definitions is that they only implicitly specify a subset of the space of all tree structures with trees that match the schema. This makes it extremely difficult to characterize how GP allocates trials to regions of the space of program trees. The difficulty

[2] The root and leaf tree fragment wildcards have slightly different meanings. A root wildcard indicates that the fragment can be embedded in some higher level subtree, while a leaf wildcard indicates possible refinement with any subtree (fragment), possibly a terminal.

is reflected in the inconclusive attempt for deriving a GP Schema Theorem based on the second schema definition presented above[O'Reilly and Oppacher, 1995].

Other definitions of GP schemata, not discussed here, also take a structural perspective and only implicitly specify what a schema stands for. The resulting theories account for GP with special operators (such as one-point crossover and mutation, see [Poli and Langdon, 1997]) or more general representations (program derivation trees, see [Whigham, 1995]).

11.3 Portraying Variable Complexity Representations

In contrast to fixed length GA representations, each member x of the GP population has a variable size s_x. The goal of this section is to define a property, the rooted-tree schema property, that makes it possible to estimate the growth in the number of individuals satisfying the property as a function of the complexity of evolved individuals, in analogy to Equation (11.1). This analysis will represent the foundation for theoretical developments, interpretations, and experiments along the following lines:

- Role of variable complexity during evolution
- Influence of parsimony penalty
- Balance between fitness/error and complexity penalty
- Alternative approaches for imposing parsimony pressure

11.3.1 The rooted-tree schema property

Generalized schemata can be defined by partitioning the space of structures with other relations. Such attempts have been presented in the GA literature [Vose and Liepins, 1991; Radcliffe, 1991]. Next we propose a simple structural property that defines a different type of partitioning of the space of programs. The space of programs will be partitioned based on the topmost structure of trees. We will call the relation induced by this property a *rooted-tree schema* or *tree-schema*.

Definition. *A* rooted-tree schema *or* tree-schema *of order k is a rooted and contiguous tree fragment specified by k function and terminal labels,* see Figure 11.1.

The definition takes advantage of the hierarchical nature of program representations. It constructively specifies the correspondence between the tree-schema representation and the subset of programs defined by the tree-schema. One can easily check if a tree belongs to a given tree-schema. The definition will allow us to capture a different view of how the space of trees is searched. It is important to note that all tree-schema of a fixed shape and order are independent and are theoretically competing for a share of the search effort.

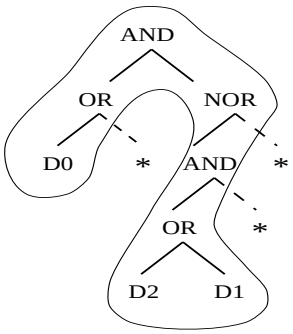

Figure 11.1
A rooted-tree fragment is the property of interest in analyzing the dynamics of GP. This example represents a tree-schema of order eight in the language used for inducing parity Boolean functions.

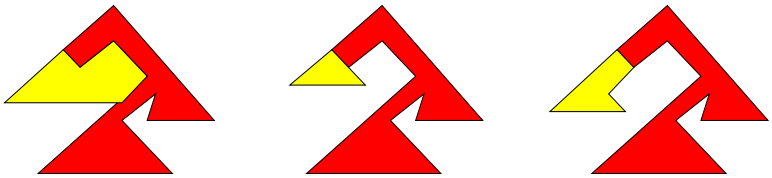

Figure 11.2
Three different tree structures defined by the same rooted-tree schema.

Note that we have dropped the implicit wild card on top of a fragment used in the previous GP schema definitions. The rooted-tree fragment can be instantiated only by refining the wild cards downwards to terminals. A rooted fragment precisely specifies, in a structural manner, the region in the space of programs to which all its instantiations will belong. It may also specify, through its composition, a collection of subtrees that are of particular importance (such as $(OR\ D_2\ D_1)$ in Figure 11.1) and how the trees are functionally combined.

There are more ways to build a tree "top," in analogy to the 2^k instantiations of k fixed bits of a GA-schema of order k. All the possible labelings of a fixed shape of the root fragment correspond to one *tree-schema class*. For any given class, the class tree-schemata compete for existence in the GP population. The larger the root fragment, the greater the number of competing tree-schemata. However, only a small fraction of them would be present in the population at any given time.

Now we can easily translate GA considerations similar to the schemata theorem or Price's theorem to GP. The distinguishing element is the variable length of chromosomes.

11.3.2 Growth of rooted-tree schemata

Consider a tree-schema $\mathcal{H}$ and the subpopulation that matches $\mathcal{H}$ (see an example in Figure 11.2.) Let $\|\mathcal{H}\|$ be the schema order[3] and $m(\mathcal{H}, t)$ be the number of copies of $\mathcal{H}$ in the population at time (generation) t. Let $f_{\mathcal{H}}$ be the average fitness of all trees matching $\mathcal{H}$ in the population and f be the average fitness of the population.[4] Consider the standard GP procedure, as defined in [Koza, 1992], that uses fitness proportionate selection for reproduction. Offspring are created through copying, tree crossover, and mutation. The probabilities for crossover and mutation are respectively p_c and p_m. Let also $p_d = p_c + p_m$.

In a population of size M, the expected number of offspring of individual $x \in \mathcal{H}$ is proportional with its fitness:

$$\frac{f_x}{\sum_{y=1}^{M} f_y} \cdot M = \frac{f_x}{f} \tag{11.2}$$

The probability of destruction for instances of individual $x \in \mathcal{H}$ is

$$p_d \cdot \frac{\|\mathcal{H}\|}{s_x} \tag{11.3}$$

By combining relations (11.2) and (11.3) we obtain a lower bound on the expected number of instances of $\mathcal{H}$ that results from offspring generated by x

$$\frac{f_x}{f} - p_d \cdot \frac{\|\mathcal{H}\|}{s_x} \cdot \frac{f_x}{f} \tag{11.4}$$

To obtain a lower bound on the expected number of copies of $\mathcal{H}$ in the next generation, we sum relations (11.4) for all trees $x \in \mathcal{H}$

$$m(\mathcal{H}, t+1) \geq \left(\sum_{x \in \mathcal{H}} \frac{f_x}{f} \right) - p_d \cdot \frac{\|\mathcal{H}\|}{f} \cdot \left(\sum_{x \in \mathcal{H}} \frac{f_x}{s_x} \right) \tag{11.5}$$

This is equivalent to

$$m(\mathcal{H}, t+1) \geq \left(\sum_{x \in \mathcal{H}} \frac{f_x}{f} \right) \left[1 - p_d \cdot \frac{\sum_{x \in \mathcal{H}} \frac{f_x}{s_x}}{\sum_{x \in \mathcal{H}} \frac{f_x}{\|\mathcal{H}\|}} \right]$$

or, taking into account that $m(\mathcal{H}, t) = \frac{1}{f_{\mathcal{H}}} \cdot \sum_{x \in \mathcal{H}} f_x$, we get

[3] The order and the defining complexity of a tree-schema schema $\mathcal{H}$ are identically defined as the number of nodes in $\mathcal{H}$, $o(\mathcal{H}) = \delta(\mathcal{H}) = \|\mathcal{H}\|$.

[4] From now on we will denote the average of quantity q over set $\mathcal{A}$ by $q_{\mathcal{A}}$ or $(q)_{\mathcal{A}}$. We also drop the index altogether when the set is the entire population. For example $s_{\mathcal{H}}$ is the average complexity of tree structures over tree-schema $\mathcal{H}$ and $\left(\frac{f}{s}\right)_{\mathcal{H}}$ is the average of ratios $\frac{f}{s}$ over $\mathcal{H}$. Similarly s is the average complexity over the entire population.

$$m(\mathcal{H}, t+1) \geq m(\mathcal{H}, t) \cdot \frac{f_{\mathcal{H}}}{f} \cdot \left[1 - p_d \cdot \underbrace{\frac{\sum_{x \in \mathcal{H}} \frac{f_x}{s_x}}{\sum_{x \in \mathcal{H}} \frac{f_x}{\|\mathcal{H}\|}}}_{a} \right] \tag{11.6}$$

Equation 11.6 accounts for the variation in the number of individuals matching tree-schema $\mathcal{H}$ due to selection and the destructive effects of crossover and mutation.[5] Let us denote as a the coefficient of p_d on the right-hand side above. It is obvious that the variation of tree-schemata is also a function of the evolved size s_x of individuals $x \in \mathcal{H}$. If all trees had the same size S ($s_x = S,\ \forall x \in \mathcal{H}$) then a would reduce to $\frac{\|\mathcal{H}\|}{S}$ which is analogous to the term combining the destructive effect of crossover and mutation in the Schema Theorem (see relation 11.1). Here, the order of the tree plays the role of defining length. When a has a very small value it does not influence the growth of above average fitness tree-schemata. A bigger value of a dampens this growth. Next we analyze in detail this dependence of the right-hand side of 11.6 on size.

11.3.3 The role of variable size during evolution

Relation 11.6 can be also written as

$$m(\mathcal{H}, t+1) \geq R_1 \cdot m(\mathcal{H}, t) \cdot \left(1 - \frac{R_2}{m(\mathcal{H}, t)} \right) \tag{11.7}$$

where R_1 and R_2 are defined as

$$R_1 = \frac{f_{\mathcal{H}}}{f}$$

$$R_2 = p_d \cdot \|\mathcal{H}\| \cdot \frac{\left(\frac{f}{s}\right)_{\mathcal{H}}}{f_{\mathcal{H}}}$$

For above average fitness tree-schemata

$$f_{\mathcal{H}} \geq f \tag{11.8}$$

i.e. $R_1 \geq 1$. We are interested in conditions under which the right-hand side of 11.7 is greater ($\geq$) or much greater ($\gg$) than $m(\mathcal{H}, t)$. Under the respective conditions, by the transitivity of $\geq$ or $\gg$, it would follow that

$$m(\mathcal{H}, t+1) \geq m(\mathcal{H}, t) \text{ or } m(\mathcal{H}, t+1) \gg m(\mathcal{H}, t) \tag{11.9}$$

[5] Fitness and complexity are time dependent too. Notations hide this time dependence for simplicity.

To determine such conditions, under which 11.9 is true, we substitute the right-hand side of equation 11.7 for $m(\mathcal{H}, t+1)$ in the first part of relation 11.9 and take into account inequality 11.8. We get

$$m(\mathcal{H}, t+1) \geq m(\mathcal{H}, t) \text{ if } m(\mathcal{H}, t) \geq \frac{R_1 R_2}{R_1 - 1} \tag{11.10}$$

After substituting R_1 and R_2, this implication can be written as

$$m(\mathcal{H}, t+1) \geq m(\mathcal{H}, t) \text{ if } m(\mathcal{H}, t) \geq \theta(\mathcal{H}, f, t) \tag{11.11}$$

where $\theta(\mathcal{H}, f, t)$ is defined as

$$\theta(\mathcal{H}, f, t) = \left(\frac{f}{s}\right)_{\mathcal{H}} \cdot \frac{1}{f_{\mathcal{H}}} \cdot \left(1 + \frac{1}{\frac{f_{\mathcal{H}}}{f} - 1}\right) \cdot p_d \cdot \|\mathcal{H}\| \tag{11.12}$$

Note that $\left(\frac{f}{s}\right)_{\mathcal{H}}$ is the average of $\frac{f}{s}$ values over the set of individuals $\mathcal{H}$. We have proved the following property:

Theorem 1. *For a given schema $\mathcal{H}$ ($\|\mathcal{H}\|$=fixed and $p_d = \text{constant}$), the number of instances of tree-schema $\mathcal{H}$ increases if $m(\mathcal{H}, t)$ exceeds a threshold value $\theta = \theta(\mathcal{H}, f, t)$ (*see relations *11.11* and *11.12).*

The interpretation is that an individual $x \in \mathcal{H}$ can increase the survival rate of $\mathcal{H}$ by decreasing the threshold factor θ in one of the following two ways (see relation 11.12):

- By increasing its fitness f_x. This would determine an increase of $f_{\mathcal{H}}$ bigger than an increase in f.
- By increasing its complexity s_x. This would determine a decrease of $\left(\frac{f}{s}\right)_{\mathcal{H}}$.

Relation 11.11 can be further analyzed as follows. The expression for $\theta = \theta(\mathcal{H}, f, t)$ can be rewritten by grouping terms that are interpretable using Chebyshev's monotonic inequality[6]

$$m(\mathcal{H}, t+1) \geq m(\mathcal{H}, t) \text{ if } m(\mathcal{H}, t) \geq \underbrace{\left(\frac{f}{s}\right)_{\mathcal{H}} \cdot \frac{s_{\mathcal{H}}}{f_{\mathcal{H}}}}_{b} \cdot \underbrace{\left(1 + \frac{1}{\frac{f_{\mathcal{H}}}{f} - 1}\right)}_{c} \cdot p_d \cdot \frac{\|\mathcal{H}\|}{s_H} \tag{11.13}$$

[6]Chebyshev's monotonic inequality states that if $a_1 \leq a_2 \leq \ldots \leq a_n$ and $b_1 \leq b_2 \leq \ldots \leq b_n$ then $(\frac{1}{n}\sum_{i=1}^{n} a_i) \cdot (\frac{1}{n}\sum_{i=1}^{n} b_i) \leq \frac{1}{n}\sum_{i=1}^{n} a_i b_i$. Also, the sense of the inequality is reversed if $b_1 \geq b_2 \geq \ldots \geq b_n$, [Graham et al., 1994].

Relation 11.13 is an equivalent form of relation 11.11 and can be interpreted as follows. Assume that $s_{\mathcal{H}} = \text{constant}$ over time steps t and $t+1$. Then the right-hand side of 11.13 depends on factors b and c. An increased value of the product $b \cdot c$ results in an increased threshold value in equation 11.13. The b factor captures the interaction between fitness and size within tree-schema $\mathcal{H}$. Let the members of $\mathcal{H}$ be indexed by $1, 2, ..., k$. If they can be ordered such that

$$s_1 \leq s_2 ... \leq s_k$$

and

$$\frac{f_1}{s_1} \leq \frac{f_2}{s_2} \leq \frac{f_k}{s_k}$$

i.e. they can be ordered such that increases in size are overcompensated by increases in fitness within $\mathcal{H}$, then by Chebyshev's inequality it would follow that $b \leq 1$. The smaller b, the smaller the threshold θ. Under the above conditions, the growth of schema $\mathcal{H}$ is favored as compared to another schema that has the same average fitness but does not satisfy the conditions. On the contrary, if fitness increases undercompensate the increases in complexity, the b component has a value bigger than 1. This determines a higher threshold value and implies that the growth of schema $\mathcal{H}$ is not equally favored.

In conclusion, this section has provided a sufficient condition for the increase in the number of instances of a tree-schema (relation 11.11) which outlines the role of variable complexity of evolved structures. By increasing in complexity but not in fitness, an individual determines a decrease in the threshold θ used in relation 11.11. If coupled with no other innovations in the population, this facilitates the increase of the individual's survival rate. For strict fitness to be dominant, fitness increases should overcompensate increases in complexity.

The increase in the survival rate of an individual solely through an increase in complexity is not a desirable tendency. Can such an effect generalize to the entire population? Section 11.6 will present simulations to answer this question. A "yes" answer is plausible, and it would indicate that the performance of GP search can be seriously degraded. The next section extends the present analysis when adding parsimony pressure during selection in order to confine the expected survival of individuals of ever increasing complexity.

11.4 Adding Parsimony

Parsimony pressure is given by an additive component $p(s_x)$ in the fitness function. Its coefficient, or weighting factor, is negative in order to penalize a size increase. With parsimony, the raw fitness function f_x is replaced by f_x^p

$$f_x^p = f_x + p(s_x) \tag{11.14}$$

A commonly used parsimony measure is linearly dependent on size

$$p(s_x) = -\sigma s_x \tag{11.15}$$

In GP practice, the choice of the weighting factor σ is very much an art. Intuitively, small positive values should be used, so that the $p(s_x)$ component is negligible compared to f_x. This can be achieved by choosing σ as follows:

$$\sigma \ll inf_x \left[\frac{f_x}{s_x}\right] \tag{11.16}$$

where infi mum is taken over a set of best solutions evolved with no parsimony pressure.

For inductive problems, we can defi ne a fi tness function that trades error and complexity. Both error and complexity are measured in information bits that have to be transmitted over a line in order to be able to recreate the original data at the other end of the line. By applying the minimum description length principle similarly to [Quinlan and Rivest, 1989], we obtain the following defi nition of the parsimony component:

$$p(s_x) = -k_1 s_x \log s_x - k_2 s_x \tag{11.17}$$

where k_1 depends on the inverse of the log of the number of fi tness cases and k_2 is proportional to the number of bits needed to encode all symbols and inversely proportional to the log of the number of fi tness cases[Rosca and Ballard, 1994].

In the rest of the article we will use a linear parsimony component.

11.4.1 Selection with a parsimonious fitness function

We compare the selection pressure in the following two cases: (1) when using a linear parsimony component (as defi ned in relations 11.14 and 11.15), and (2) when using "plain" fi tness. The result is synthesized by the following theorem:

Theorem 2. *With linear parsimony penalty, individuals x with a fitness-complexity ratio greater than the ratio of population averages f and s:*

$$\frac{f_x}{s_x} \geq \frac{f}{s} \tag{11.18}$$

have greater selection likelihood. This property is true regardless of the value of σ.

Proof: The role of parsimony can be formally analyzed by comparing the expected number of offspring of an individual x in the two cases above ("$\vee$" below means "versus"):

$$\frac{f_x - \sigma s_x}{\sum_y (f_y - \sigma s_y)} \cdot M \vee \frac{f_x}{\sum_y f_y} \cdot M \tag{11.19}$$

Relation 11.18 is obtained after truth preserving simplifi cations in 11.19. □

Theorem 2 allows one to interpret whether selection is stronger with or without parsimony. With parsimony, individuals x with a fitness-complexity ratio greater than the ratio of population averages f and s have greater selection likelihood. Although σ is factored out relation 11.18, it influences how much bigger the selection pressure is and what gets selected. A stronger selection pressure towards more effective individuals is useful in early stages of evolution and may be undesirable in later stages due to the decreased population diversity and thus stronger convergence tendency towards a local optimum.

11.4.2 Growth of rooted-tree schemata with parsimony

When a parsimonious fitness function is used, f is replaced by f^p, which is defined here according to relations 11.14 and 11.15.[7] Let θ^p be the value of θ obtained by substituting f^p for f in equation 11.12. The survival rates of a schema in the two cases can be compared by comparing the value of θ^p with θ.

The direct comparison $\theta \vee \theta^p$ can be written as

$$\left(\frac{f}{s}\right)_{\mathcal{H}} \cdot \frac{1}{f_{\mathcal{H}} - f} \vee \left(\frac{f-\sigma s}{s}\right)_{\mathcal{H}} \cdot \frac{1}{(f-\sigma s)_{\mathcal{H}} - (f-\sigma s)} \tag{11.20}$$

The average operator over $\mathcal{H}$, implicit in the average notation $(\cdot)_{\mathcal{H}}$, has the following properties that are used to simplify equation 11.20:

$$\left(\frac{f-\sigma s}{s}\right)_{\mathcal{H}} = \left(\frac{f}{s}\right)_{\mathcal{H}} - \sigma$$

$$(f-\sigma s)_{\mathcal{H}} = f_{\mathcal{H}} - \sigma s_{\mathcal{H}}$$

After simplifications, $\theta \vee \theta^p$ becomes equivalent to

$$\left(\frac{f}{s}\right)_{\mathcal{H}} \cdot \frac{1}{f_{\mathcal{H}} - f} \vee \left[\left(\frac{f}{s}\right)_{\mathcal{H}} - \sigma\right] \cdot \frac{1}{(f_{\mathcal{H}} - f) - \sigma(s_{\mathcal{H}} - s)} \tag{11.21}$$

When complexity $s_{\mathcal{H}}$ increases the left-hand side of 11.21 decreases. However, this is not necessarily true with the right-hand side, where the term

$$\frac{1}{(f_{\mathcal{H}} - f) - \sigma(s_{\mathcal{H}} - s)}$$

increases with $s_{\mathcal{H}}$. As expected, this shows that size plays a different role in the dynamics of GP with parsimony pressure. In this case it is obvious that selection will favor the smaller of two evolved structures of equal raw fitness

$$s_{x_1} \leq s_{x_2} \Leftrightarrow f^p_{x_1} \leq f^p_{x_2} \tag{11.22}$$

[7] So far we have bypassed details about the nature of f, such as whether f is raw fitness or normalized fitness (see [Koza, 1992]), and rather considered that the fitness function f supplies values used during the selection phase of the GP algorithm.

This implies size decreases over periods of time when fitness does not improve.

When $\mathcal{H}$ identifies a worthy individual x just discovered, relation 11.20 can be simplified. The worthy structure is propagated throughout the population at a higher rate if the threshold θ is much lower than the threshold with parsimony θ^p. This is true if and only if the right hand side below holds

$$\theta \gg \theta^p \Leftrightarrow \frac{f_x}{s_x} \gg \frac{f}{s} \tag{11.23}$$

The following theorem covers the general case:

Theorem 3. *For an above average schema $\mathcal{H}$, if σ is chosen as in relation 11.16 and the relative increase in schema fitness is bigger than the relative increase in schema size times the average of ratios $\left(\frac{f}{s}\right)_{\mathcal{H}}$, then a better than average schema has better chances of surviving and propagating in the population. More precisely*

$$\theta \gg \theta^p \Leftrightarrow (f_{\mathcal{H}} - f) \gg \left(\frac{f}{s}\right)_{\mathcal{H}} \cdot (s_{\mathcal{H}} - s) \tag{11.24}$$

The above conclusions are consistent with the analysis of the role of size in selection and can be particularized for the case $\mathcal{H} = \{x\}$ to reach the same conclusion as in 11.23.

The size increase tendency is deamplified, as expected, when a parsimonious fitness function is considered versus the case of the tree-schema growth in equation 11.11. However in demarcation situations, such as the discovery and proliferation of a fit structure, size influences its rate of survival as shown in relation 11.23. In section 11.6 we will explore the role of size in GP simulations where a parsimonious fitness function is considered.

11.5 Controlling Schema Growth

The probability of destruction of instances of individual $x \in \mathcal{H}$ depends on the complexity s_x (see equation 11.3). In one likely scenario the complexity of the current best programs tends to increase. Also, the proportion of the more complex of these programs in the population increases although they do not improve in raw performance. This would make GP prone to local minima and result in a waste of computational effort. The result is comprised in the growth of tree-schemata in relation 11.11 and the interpretation of Theorem 1. GP search is unjustifiably biased towards exploring structures of higher and higher complexity independently of raw fitness.

On the contrary, when using parsimony the above problem is apparently accounted for. This is done at the price of modifying the fitness function to penalize for increases in complexity. However, search optima with the new fitness function are not necessarily search optima with the original fitness function. In other words, the fitness "landscape" changes.

It would be desirable to inhibit the propagation of structures that increase in complexity without improving in fitness. The remedy to this problem is to control tree schema growth so that the complexity of structures does not influence the probability of tree-schema destruction. In other words, GP should not allow overall size growth to "protect" a rooted-tree schema from disruption. This can be achieved in two ways.

First, such an effect could be approximately obtained by predefining a probability mass function for choosing crossover and mutation points within tree structures, which would assign most of the probability mass to the higher tree levels, i.e. to nodes closer to the root. Figure 11.3 gives an example of the desired shape of the probability mass function, which is chosen to be the Pascal (negative binomial) distribution. The depth of the point of variation is chosen based on this probability mass function, and the node itself is chosen uniformly among all nodes of the same depth. If a depth is not represented in some tree structure, that structure is copied unchanged. A tree structure grows downwards, so that the cumulative probability of disrupting a given number of the high layers in the tree becomes independent of the tree size. Therefore, this method achieves the goal of disrupting a tree-schema with constant, complexity independent, probability. Unfortunately, the method introduces the parameters of the probability mass function that would have to be eventually adjusted (see Figure 11.3). Another practical disadvantage is the intricacy of any method for determining the actual crossover/mutation point.

Second, we could directly affect the probabilities of crossover and mutation, i.e. make them adapt, so that disruption of tree-schemata occurs with probability independent of complexity. The basic idea is to redefine the probability of disruption of structure x to be a function of s_x

$$p'_d = \begin{cases} p_d & \text{if } s_x < s_0 \\ \min\{1, p_d \cdot \frac{s_x}{s_0}\} & \text{if } s_x \geq s_0 \end{cases} \tag{11.25}$$

where s_0 is a predefined parameter playing the role of a threshold complexity value (see Figure 11.4). With this change, disruption of schema $\mathcal{H}$ occurs with constant probability when $s_x \geq s_0$

$$p'_d \cdot \frac{\|\mathcal{H}\|}{s_x} = p_d \cdot \frac{\|\mathcal{H}\|}{s_0} = \text{constant} \tag{11.26}$$

If s_x increases then the probability of destruction of x remains constant. The number of instances of x in the next generation will be exclusively influenced by its fitness and the appearance in the population of better individuals.

11.6 Experimental Results

This section presents experiments aimed at reinforcing conclusions that have been suggested by the previous theoretical analysis.

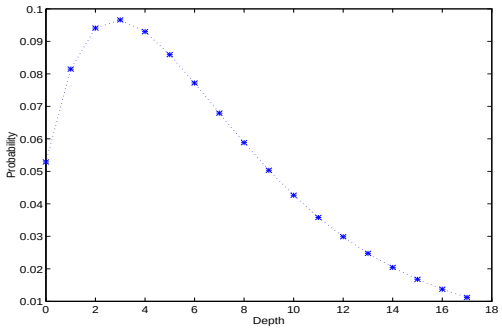

Figure 11.3
Depth of variation points is chosen using a negative binomial distribution with $r = 2$ and $p = 0.23$. The cumulative distribution up to a depth of trees of 17 is 0.999.

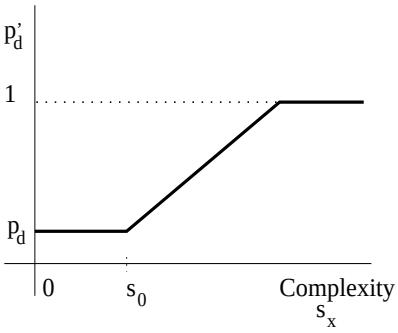

Figure 11.4
Rule for the automatic adaptation of the probability of disruption p'_d. Probabilities p_m and p_c can be updated proportionately so that their sum is p'_d.

One cannot know in advance what particular schema is to be preferred by the evolutionary process in a run of GP. Moreover, it is extremely hard to trace a schema property for all competing tree-schemata of any given shape and size. However, one can examine relevant properties globally, for the entire population, and narrowly, for the best individual in the population. The experiments below provide a consistent view of GP dynamics by looking at a common set of measures over three types of runs of the standard GP engine. The measures are quantities that have appeared throughout our derivations and have been used to qualitatively explain results. They are the averages over the population (f, s, $\frac{f}{s}$) and the best individual in a generation f_{best}, s_{best}, or $\left(\frac{f}{s}\right)_{best}$. The types of runs performed are:

1. Standard GP with raw fitness being only a measure of performance.

2. Standard GP with parsimony pressure. The fitness function combines raw fitness and a linear parsimony component to penalize a size increase.

3. GP with adaptive probabilities of crossover and mutation. These probabilities are updated dynamically as in Figure 11.4 in order to impose a constant parsimony pressure on competing tree-schemata regardless of the complexity of evolved structures.

Two test problems are used. The first problem is the induction of a Boolean formula (circuit) that computes parity on a set of bits [Koza, 1994]. The raw fitness function has access to fitness cases that show the parity value for all inputs, and counts the number of correct parity computations. These experiments use the following parameters: population

size $M = 4000$, number of generations $N = 50$, crossover rate $p_c = 89\%$ (20% on leaf nodes), mutation rate $p_m = 1\%$, reproduction rate $p_r = 10\%$, number of fitness cases = 2^n where n is the order of the parity problem. The second problem is the induction of a controller for a robotic agent in a dynamic and nondeterministic environment, as in the Pac-Man game [Koza, 1992; Rosca and Ballard, 1996a]. Raw fitness here is computed from the performance of evolved controllers over a number of simulations. The parameters for these runs are $M = 500$, $N = 150$, $p_c = 89\%$, $p_m = 1\%$, $p_r = 10\%$, with three simulations determining the individual fitness. Also, $s_0 = 100$.

Each of Figures 11.5-11.7 contains three plots: (a) Variation of average complexity s and the complexity of the best-of-generation individual s_{best} (top); (b) Variation of the ratio of averages $\frac{f}{s}$ and of $\left(\frac{f}{s}\right)_{best}$ (middle); (c) The fitness learning curve f_{best} and variation of average fitness f (bottom).

11.6.1 Fitness based on pure performance

Over the time span of evolution in a GP run, often there are long periods of time when no fitness improvements are noticed. Section 11.3.2 has proved that an increase of the individual's survival rate can be accomplished by the increase in its complexity, but not in its fitness. Can this effect generalize to the entire population? We suggested that a "yes" answer is plausible, which indicates that the performance of the GP engine can be seriously degraded. Here we present experimental evidence.

The variation in the complexity of evolved structures can be seen in plots correlating the learning curves and complexity curves for the two test problems. When fitness remains constant, both the best of generation complexity s_{best} and the average complexity s indeed increase over time. Plateaus of f_{best} can be observed in Figure 11.5(c) (left) between generations 33 and 59, or Figure 11.5(c) (right) between generations 15 and 47, or 53 and 101. During the corresponding time intervals, size almost doubles in Figure 11.5(a) (left) and significantly increases in Figure 11.5(a) (right) while average fitness also increases. The increase in average size is explained by the predominant increase in survival rate of above average individuals of increased size in the absence of any fitness improvements.

11.6.2 Parsimonious fitness

Next we present experiments where parsimony pressure is applied during selection in order to confine the survival of individuals of ever increasing complexity and thus to guard against the apparent loss of efficiency of GP search. An important question is whether parsimony would deter GP search from finding fit solutions at the expense of finding parsimonious solutions. This could happen because of the artificial distortion in fitness created by the parsimony component.

We expect to see a decrease in size over spans of time with no improvement in fitness. Three such intervals can be noticed in Figure 11.6(a) and (c) (left): generations 18 to

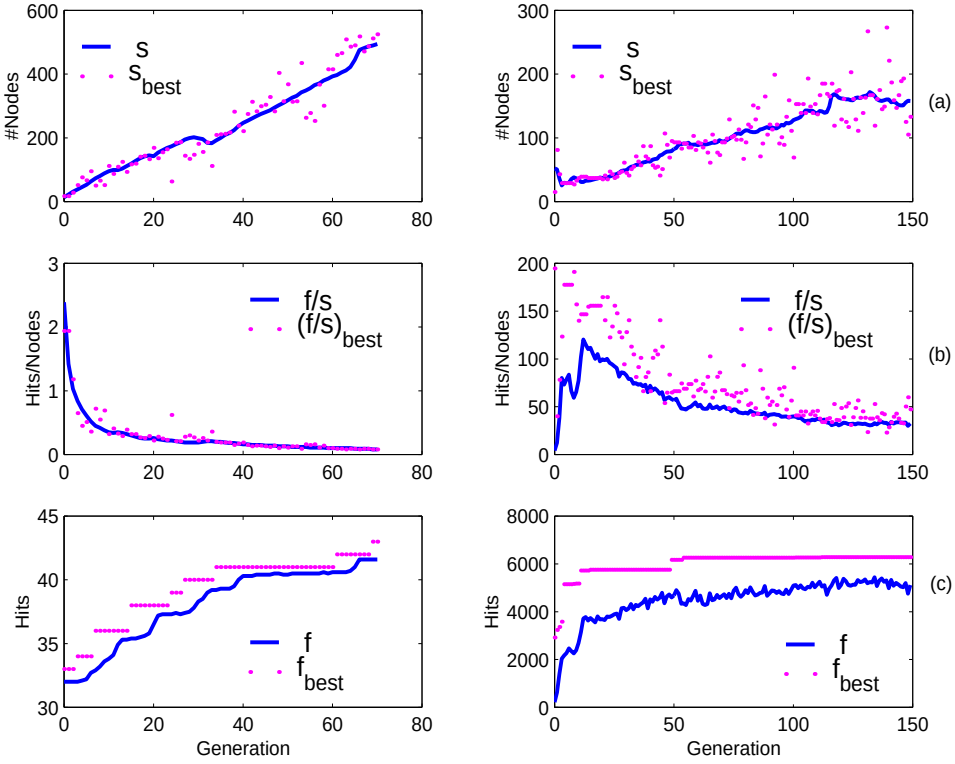

Figure 11.5
Average and best-of-generation variation in (a) size, (b) fitness-size ratio, and (c) fitness in a run of GP without parsimony pressure: Parity (left) and Pac-Man (right).

57, 58 to 72, and 73 to 99. The GP algorithm discovers a new best solution, having a complexity higher than every other previous individual at the beginning of each of these intervals. Following the complexity curves towards the end of the intervals, we note a gradual decrease in s_{best}. The same tendency is conspicuous in the average complexity plot s, which has a shape similar to s_{best} and is delayed with about four to five generations. The delay period is the time needed by selection to pick up on the opportunities created at the beginning of the above intervals.

One remarkable feature of the f plot in Figure 11.6(c) is that average fitness decrease is correlated with average size decrease s. The explanation is that parsimony pressure induces a decrease in complexity, which makes mutation and crossover operations more disruptive. This generates a decrease in average fitness over the population. The effect is even clearer when the value of the weighting factor σ increases (see Figure 11.6) (right).

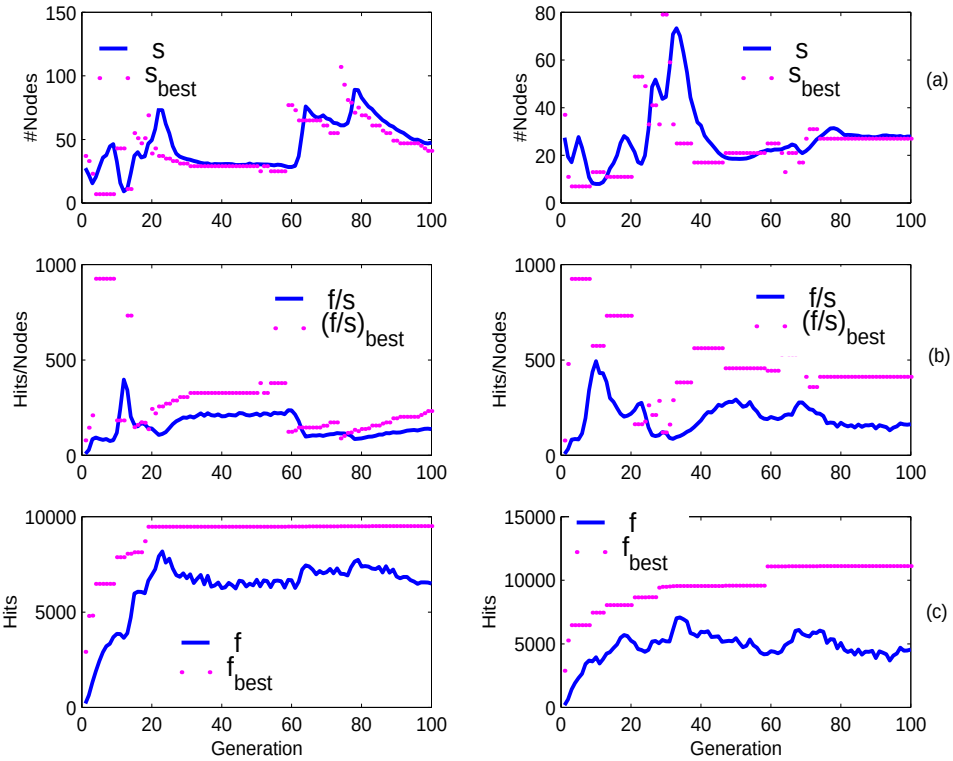

Figure 11.6
Average and best-of-generation variation in (a) size, (b) fitness-size ratio, and (c) fitness in a run of GP with parsimony pressure $\sigma = 0.1$ (left) and $\sigma = 1.0$ (right) for the Pac-Man problem.

Note also the rapid increases in fitness in early generations in Figure 11.6(b) (left) and (right). They show that the following relation holds in very early generations in contrast to Figure 11.5(b) (left) and (right).

$$\frac{f_{best}}{s_{best}} \gg \frac{f}{s}$$

i.e. the stronger selection pressure towards more effective individuals due to parsimony is useful to rapidly focus search towards good structures (as in the discussion of equations 11.18 and 11.23). The ability of the GP engine to find fit solutions appears to have improved considerably when using a parsimonious fitness function.

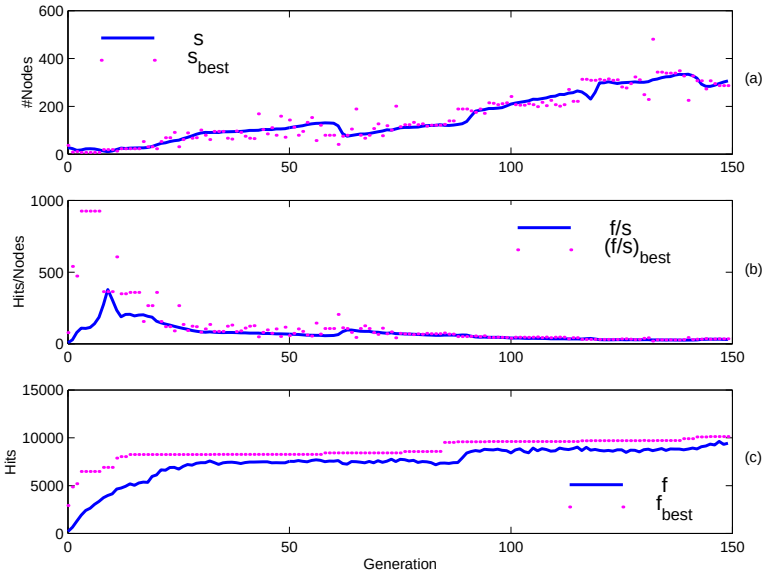

Figure 11.7
Average and best-of-generation variation in (a) size, (b) fitness-size ratio, and (c) fitness in a Pac-Man run of GP with autoadaptive crossover, mutation, and reproduction rates.

11.6.3 Adaptive probability of destruction

In this third experiment we modify the standard GP engine to consider an adaptive probability of disruption of an expression (mutation and crossover) dependent on the size of the expression, as in equation 11.25. The standard GP procedure varies selected subexpressions in proportion to p_c and p_m and keeps around (in next generation) surviving selected structures in proportion to $p_r = 1 - p_c - p_m$. This is done globally in the sense that a p_c fraction of the next generation is obtained through crossover on selected structures, etc. In contrast, the size-adaptive procedure decides what genetic operation to apply for each selected individual. In this way, the complexity of the individual can be used in taking the decision of mutation, crossover or survival. The procedure records the proportions of the next generation obtained with each genetic operation.

An example is given in Figure 11.8 where one can see the variations in the probability of crossover (the only type of variation used). The changes in p_c can be correlated with the changes in the average complexity s (Figure 11.7). Although size increases over time, the higher destruction appears to limit the size increase without disrupting the search process.

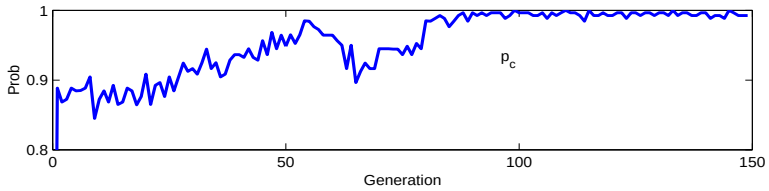

Figure 11.8
Adaptation in the probability of crossover for the run in Figure 11.7.

11.6.4 Summary of experiments

The experiments above have attained three main goals in relation with the theoretical analysis of tree-schema growth:

• Traced the GP-specific variable complexity during evolution and interpreted its variations from the perspective of the size-dependent growth formula 11.11. Complexity increases can derail the search effort of the GP engine. This is showed by the long stable periods with no improvements in the most fit structures.

• Traced the influence of an additive parsimony component to fitness. Experiments show a considerable increase in efficiency and offer insight into the choice for the value of σ, the weighting factor of the parsimony component.

• Observed at work the second proposed alternative to imposing parsimony pressure. By increasing the probability of variation while the size of expressions increases, we preserve the exploration ability of GP in the space of expressions. More experiments are needed to assess the advantage of this adaptive method.

11.7 Solution Acquisition in GP

It is important to understand how partial solutions are refined, knowing that GP does not merely guess solutions. Such an understanding would facilitate the design of tools to make the evolutionary process more transparent and moreover to control it. Efforts along this line have been recently reported in [Goldberg and O'Reilly, 1998; O'Reilly and Goldberg, 1998], who looked at how independent sub-solutions are acquired as a function of their fitness contributions. Here, we are rather interested in the structural aspect of solution construction, when it is unknown what constitutes a subsolution.

The classical view, in line with the building block hypothesis in GA/GP, is that GP solutions are refined *bottom-up*, i.e. by starting with small pieces of useful code and then by combining them to compose higher and higher order components. Such pieces of code

should be protected from crossover and other disruptive genetic operators [Banzhaf et al., 1998]. There is little evidence of what building blocks actually are in this case.

The GP representation using Automatically Defined Functions [Koza, 1992] makes unambiguous what could possibly be building blocks, namely subroutines. Subroutines can be hierarchically invoked in order to build solutions, and are also subject to genetic variations. The bottom-up view appears justified when using subroutines. ADFs or subroutines should be *stable* building blocks. Experimental evidence that they are indeed refined bottom-up was presented in [Rosca, 1995]. [Rosca, 1995] considered representations with three components: two subroutines and the main program body. It computed the percentage of genetic changes in a component, taken over all programs in the genealogic tree of a solution created by GP, as a function of generation. The analysis showed that the percentage of changes in a component (such as an ADF) achieves a maximum at times directly proportional to the place of the component in the component hierarchy. Thus, components at the bottom of the hierarchy (such as ADF_0) are learned first, and are subsequently used to define higher level components.

The rooted-three schema theory suggests a radically different view, namely that solutions are refined *top-down*, starting with the root of the tree and specializing the problem decomposition in a top-down manner. Indeed, a small number of independent rooted-tree schemata compete for existence in the population at any given time. A rooted-tree schema that wins the competition is further refined downwards (i.e top-down). Equivalently, the competition within a certain schema order ends (with one rooted-tree schema being a winner). It gradually transforms into a competition among independent rooted-tree schemata descendents (of the winning schema) having higher order. Two rooted-tree schemata are different if and only if they diverge in shape or, if they have the same shape, they have at least one different point. Different rooted-tree schemata are independent, i.e. they refer to (syntactically) disjoint regions of the search space (as opposed to GA schemata, which may not be independent even when they are different).

The top-down refinement hypothesis advanced here is consistent with the observation that crossover acts as a "macromutation." In other words, simply using a mutation operator that replaces a randomly chosen subtree with a new randomly created subtree results in at least similar performance to GP with crossover [Angeline, 1997].

Although the top-down and bottom-up views are radically different, they are not mutually exclusive. Top-down refinement may lead to program components that are effective and essential in any fitness computation. Such components are still evolved bottom-up, nonetheless the overall process of solution refinement is top-down, as evidenced in the experiments from [Rosca, 1995] (see a detailed account in [Rosca, 1997]). A related idea informally states that "two types of building blocks are used to construct solutions" [Poli and Langdon, 1997, page 284]. These would correspond to rooted-tree fragments and subtree fragments.

11.8 Related Work

The GP literature is rich in deliberations on the bloat phenomenon (size increase), and bloat control schemes. Herein, we recapitulate a few and place them in a broader perspective.

Early, [Tackett, 1994] pointed out that bloating cannot be selection-neutral. He presented experiments suggesting that average growth in size is proportional to the selection pressure. In our analysis, selection pressure itself is complexity dependent. Tackett also suggested that the larger programs selected by GP contain expressions which are inert overall (introns), but contain useful subexpressions, thus correlating bloating with hitchhiking.

Related to the size problem, GP research has focused on the analysis of introns. *Introns* are pieces of code with no effect on the output. An analysis of introns goes hand in hand with an analysis of bloating. [Nordin et al., 1995] tracked introns in an assembly language GP system based on a linear but variable length program representation. The analysis suggested that the increase in size is a "defense against crossover." A similar conclusion is reached here in Theorem 1 (Section 11.11). In the linear representation, the noticed increase in the size of programs was attributed to introns. Based on experiments with controlled crossover or mutation rate within intron fragments, [Nordin et al., 1995] suggested that a representation which generates introns leads to better search effectiveness. Thus, introns may have a positive role in GP search protecting against destructive genetic operations. For hierarchical GP representations [Rosca, 1996] showed that much of the size increase is due to ineffective code too. However, the role of introns has been disputed in the case of GP using tree representations [Andre and Teller, 1996]. For one thing, the overhead introduced by exponentially increasing tree sizes may offset any protective effects of introns.

More recently, the GP literature contains several puzzlingly simple and informal attempts to explaining the bloat phenomenon. [Langdon and Poli, 1997] claims that larger, semantically equivalent programs are more abundant, therefore they are more likely to be found by GP search. However, non-equivalent larger programs are much more abundant. Therefore, it is unclear whether the proportion of semantically equivalent larger programs increases with the depth or size of programs. [Soule and Foster, 1998] outlines a processing bias called "removal bias" of standard GP operators. The replacement of small subtrees with bigger ones is more likely not to affect offspring, therefore size tends to increase. This explanation agrees with the increase in survivability of offspring predicted by rooted-tree schemata.

A diversity of bloat control schemes have been proposed in the literature. These vary from complexity bounds and penalties, to submodule reuse, biased genetic operations to counteract the bloat bias, and multi-objective or co-evolving fitness.

The problem of learning a non-parametric model without biasing for particular structures has been addressed in the area of nonparametric statistical inference. In statistical terms this is the problem of learning with low bias or tabula-rasa learning. However, low bias in the choice of models is paid for by a high variance (see [Geman et al., 1992] for

an excellent introduction to the bias/variance dilemma). Methods for balancing bias and variance include techniques that rely on a complexity penalty function which is added to the error term in order to promote parsimonious solutions. The basic idea is to trade the complexity of the model for its accuracy. This idea resonates with one of the fundamental principles in inductive learning represented by Ockham's razor principle, which is interpreted as: "Among the several theories that are consistent with the observed phenomena, one should pick the simplest theory" [Li and Vitanyi, 1992]. What is simple often turns out to be more general.

One common approach to dealing with a variable complexity model within the Bayesian estimation framework is Rissanen's minimum description length (MDL) principle [Li and Vitanyi, 1993]. The MDL principle trades off the model code length, i.e. the complexity term, against the error code length, i.e. the data not explained by the model or error term. Complexity is naturally expressed as the size of code or data in bits of information.

Informed approaches to including a parsimony component, such as the MDL principle, implicitly expect that the capability of the learned model is a smooth function of its complexity, i.e. as a solution grows it is fitter. This is not true for Genetic Programming, which furthermore cannot afford to exploit a large number of training examples and use infinite populations in order to overcome the problem. A small change in a program can entirely destroy its performance. The capability of a model specified with a program is not a smooth function of its complexity. Nonetheless GP manages to sample the space of programs and to discover automatically satisfiable models of variable complexity.

The MDL principle has been also applied in GP to extend the fitness function of hybrid classification models [Iba et al., 1993; Iba et al., 1994]. For example [Iba et al., 1994] applied the MDL principle in the learning rule of a GP-regression tree hybrid. [Zhang and Muhlenbein, 1995] used an adaptive parsimony strategy in a GP-neural net hybrid. In both cases GP manipulates tree structures corresponding to a hierarchical multiple regression model of variable complexity, decision trees, or sigma-pi neural networks, rather than programs. MDL-based fitness functions have been unsuccessful in the case of GP evolving pure program structures. Iba outlined that the MDL-based fitness measure can be applied problems satisfying the "size-based performance" criterion [Iba et al., 1994], where the more the tree structure grows the better its performance becomes.[Rosca and Ballard, 1994] has used the MDL principle to assess the suitability of an extension of GP with subroutines called adaptive representation (AR).

The most common approach to circumvent complexity-induced limitations in GP has been the use of a parsimonious fitness function. Parsimony imposes constraints on the complexity of learned solutions. However the effects of such constraints in GP have not been elucidated. Parsimony pressure clearly improves efficiency of search and understandability of solutions if well designed. The quality and, in particular, the generality of solutions may also be improved in inductive problems.

However, adding the right parsimony pressure has been more of an art. One example of avoiding this decision by means of an ad-hoc algorithm is "disassortative mating" [Ryan,

1994]. This GP algorithm selects parents for crossover from two different lists of individuals. One list of individuals is ranked based on fitness while the other is ranked based on the sum of size and weighted fitness. The goal is to evolve solutions of minimal size that solve the problem. However, it was recognized that by directly using the size constraint the GP algorithm is prevented from finding solutions. The disassortative mating algorithm is reported to improve convergence to a better optimum while maintaining speed.

Another suggestion for limiting the increase in complexity is to employ modular GP extensions such as algorithms based on the evolution of the architecture [Koza, 1994], module acquisition [Angeline, 1993], heuristic extensions for the discovery of subroutines [Rosca and Ballard, 1994; Rosca and Ballard, 1996a], or GP with architecture modifying operation using code duplication [Koza, 1995]. Evolved modular programs theoretically have a lower descriptional complexity [Rosca and Ballard, 1994] and also appear to present better generality [Rosca, 1996; Rosca and Ballard, 1996b].

The problem that evolved expressions tend to drift towards large and slow forms without necessarily improving the results was recognized in some excellent early work in GP applied to the simulation of textures for use in computer graphics [Sims, 1991]. The solution devised was heuristic. Mutation frequencies were tailored so that a decrease in complexity was slightly more probable than an increase. This did not prevent increases towards larger complexity but more complex solutions were due to the selection of improvements. It is not apparent how this was accomplished precisely. Interestingly, the solution to controlling complexity presented in section 11.5 achieves exactly this effect and is theoretically founded.

An alternative approach to program induction based on different search principles is taken in ADATE [Olsson, 1995]. ADATE essentially performs iterative deepening, exhaustive search in the space of program constructs, and it proves quite successful for small target programs. Interestingly enough from the perspective of how GP assembles solutions, ADATE refines solutions in a top-down manner by design.

11.9 Conclusion

Researchers from machine learning in general and evolutionary computation in particular have long debated the utility of the schema theory originated by Holland in the mid-seventies as a mathematical tool for characterizing the mechanism of a genetic algorithm. This chapter addressed the issue of finding a relevant property of GP evolved structures whose analysis, inspired in the GA schema theory, could provide insight into how GP works and reveal possible search limitations caused to the standard GP engine by the variable complexity of evolved structures.

The property of interest, called *rooted-tree schema*, is defined as a relation on the space of program structures. Rooted-tree schemata provide a partitioning of the GP search space and show how GP allocates effort to *independent* regions of the search space described

by independent competing rooted-tree schemata. The analysis of rooted-tree schemata is remarkably simple and powerful.

Formal analysis of rooted-tree schemata makes it possible to understand the role played by variable complexity of evolved structures. The analysis showed the influence of a parsimony component and discussed choices on the complexity weighting factor values. Alternative approaches to parsimony pressure were proposed, based on the idea of varying the probability of destructive genetic operations proportionally to the complexity of structures. Last but not least, the rooted-tree schema theory suggests a radically different view of program refinement in GP, namely that solutions are refined top-down starting with the root of the tree. This observation is important from the perspective of defining more powerful genetic operators and second tier control heuristics of the GP system itself. Theoretical results were also interpreted and discussed from an experimental perspective.

The rooted-tree schema analysis reveals insights about the mechanisms at work in GP, and should be of interest to a machine learning audience eager to find formal arguments in addition to pure experimental evidence from the field of genetic programming.

Acknowledgements

Una-May deserves sincere thanks for her careful reading and excellent suggestions for improving this chapter. Bill deserves many thanks for being such a terrific colleague.

Bibliography

Altenberg, L. (1995), "The schema theorem and Price's theorem," in *Foundations of Genetic Algorithms 3*, L. D. Whitley and M. D. Vose (Eds.), pp 23–49, San Mateo, CA, USA: Morgan Kaufmann.

Andre, D. and Teller, A. (1996), "A study in program response and the negative effects of introns in genetic programming," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), pp 12–20, The MIT Press.

Angeline, P. J. (1993), *Evolutionary Algorithms and Emergent Intelligence*, PhD thesis, Ohio State University.

Angeline, P. J. (1994), "Genetic programming and emergent intelligence," in *Advances in Genetic Programming*, K. E. Kinnear, Jr. (Ed.), Chapter 4, pp 75–98, Cambridge, MA, USA: MIT Press.

Angeline, P. J. (1997), "Subtree crossover: Building block engine or macromutation?," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), pp 9–17, Stanford University, CA, USA: Morgan Kaufmann.

Bäck, T., Hoffmeister, F., and Schwefel, H.-P. (1991), "A survey of evolutionary strategies," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, Inc.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998), *Genetic programming: An Introduction*, Morgan Kaufmann and dpunkt.

Fogel, D. B. (1995), *Evolutionary computation : toward a new philosophy of machine intelligence*, IEEE Press.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966), *Artificial intelligence through simulated evolution*, Wiley.

Geman, S., Bienenstock, E., and Doursat, R. (1992), "Neural networks and the bias/variance dilemma," *Neural Computation*, (4):1–58.

Goldberg, D. and O'Reilly, U.-M. (1998), "Where Does the Good Stuff Go, and Why? How contextual semantics influences program structure in simple genetic programming," in *Proceedings of the First European Workshop on Genetic Programming*, W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (Eds.), volume 1391 of *LNCS*, Paris: Springer-Verlag.

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

Graham, R. L., Knuth, D. E., and Patashnik, O. (1994), *Concrete mathematics : a foundation for computer science*, Addison-Wesley, 2nd edition.

Grefenstette, J. and Baker, J. (1989), "How genetic algorithms work: a critical look at implicit parallelism," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, J. Schaffer (Ed.), Morgan Kaufmann.

Holland, J. H. (1992), *Adaptation in Natural and Artificial Systems, An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*, Cambridge, MA: MIT Press, Second edition (First edition, 1975).

Iba, H., de Garis, H., and Sato, T. (1994), "Genetic programming using a minimum description length principle," in *Advances in Genetic Programming*, K. Kinnear Jr. (Ed.), MIT Press.

Iba, H., Kurita, T., de Garis, H., and Sato, T. (1993), "System identification using structured genetic algorithms," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, Pittsburg, PA, USA: Morgan Kaufmann Publishers, Inc.

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

Koza, J. R. (1994), *Genetic Programming II*, MIT Press.

Koza, J. R. (1995), "Gene duplication to enable genetic programming to concurrently evolve both the architecture and work-performing steps of a computer program," in *IJCAI*, C. S. Mellish (Ed.), volume 1, pp 734–740, Morgan Kaufmann.

Langdon, W. B. and Poli, R. (1997), "Fitness causes bloat," in *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*.

Langdon, W. B. and Poli, R. (1998), "Fitness causes bloat: Mutation," in *Proceedings of the First European Workshop on Genetic Programming*, W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (Eds.), volume 1391 of *LNCS*, pp 37–48.

Li, M. and Vitanyi, P. (1993), *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag.

Li, M. and Vitanyi, P. M. B. (1992), "Inductive reasonong and kolmogorov complexity," *Journal of Computer and Systems Sciences*, 44:343–384.

Mühlenbein, H. (1991), "Evolution in time and space – the parallel genetic algorithm," in *Foundations of Genetic Algorithms 1*, G. J. Rawlins (Ed.), pp 316–337, Morgan Kaufmann.

Mitchell, M. (1996), *An Introduction To Genetic Algorithms*, MIT Press.

Nordin, P., Francone, F., and Banzhaf, W. (1995), "Explicitly defined introns and destructive crossover in genetic programming," in *Proceeedings of the ICML Workshop on Genetic Programming: From Theory to Real-World Applications (NRL TR 95.2)*, J. P. Rosca (Ed.), pp 6–22, University of Rochester.

Olsson, R. (1995), "Inductive functional programming using incremental program transformation," *Artificial Intelligence*, 74:55–81.

O'Reilly, U.-M. and Goldberg, D. (1998), "How fitness structure affects subsolution acquisition in genetic programming," in *Proceedings of the Third Annual Genetic Programming Conference*, 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.), Madison, WI, USA: Morgan Kaufmann.

O'Reilly, U.-M. and Oppacher, F. (1995), "The troubling aspects of a building block hypothesis for genetic programming," in *Foundations of Genetic Algorithms 3*, L. D. Whitley and M. D. Vose (Eds.), pp 73–88, San Mateo, CA, USA: Morgan Kaufmann.

Poli, R. and Langdon, W. B. (1997), "A new schema theory for GP with one-point crossover and point mutation," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), pp 278–285, Stanford University, CA, USA: Morgan Kaufmann.

Price (1970), "Selection and covariance," *Nature*, 227:520–521.

Quinlan, J. R. and Rivest, R. L. (1989), "Inferring decision trees using the minimum description length principle," *Information and Computation*, pp 227–248.

Radcliffe, N. J. (1991), "Equivalence class analysis of genetic algorithms," *Complex Systems 5*, (2):183–205.

Radcliffe, N. J. (1992), "Non-linear genetics representations," in *Parallel Problem Solving from Nature, 2*, R. Männer and B. Manderick (Eds.), Elsevier Science Publishers.

Rosca, J. P. (1995), "Genetic programming exploratory power and the discovery of functions," in *Evolutionary Programming IV Proceedings of the Fourth Annual Conference on Evolutionary Programming*, J. R. McDonnell, R. G. Reynolds, and D. B. Fogel (Eds.), pp 719–736, San Diego, CA, USA: MIT Press.

Rosca, J. P. (1996), "Generality versus size in genetic programming," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), pp 381–387, Cambridge, MA: The MIT Press.

Rosca, J. P. (1997), *Hierarchical Learning with Procedural Abstraction Mechanisms*, PhD thesis, University of Rochester, Rochester, NY 14627.

Rosca, J. P. and Ballard, D. H. (1994), "Hierarchical self-organization in genetic programming," in *11th International Conference on Machine Learning*, pp 251–258, Morgan Kaufmann.

Rosca, J. P. and Ballard, D. H. (1996a), "Discovery of subroutines in genetic programming," in *Advances in Genetic Programming 2*, P. Angeline and K. E. Kinnear, Jr. (Eds.), Chapter 9, Cambridge, MA, USA: MIT Press.

Rosca, J. P. and Ballard, D. H. (1996b), "Evolution-based discovery of hierarchical behaviors," in *Proc. of the Thirteenth National Conference on Artifi cial Intelligence (AAAI-96)*, pp 888–894, AAAI Press/The MIT Press.

Ryan, C. O. (1994), "Pygmies and civil servants," in *Advances in Genetic Programming*, K. E. Kinnear, Jr. (Ed.), MIT Press.

Sims, K. (1991), "Artifi cial evolution for computer graphics," *Computer Graphics*, 25(4):319–328.

Soule, T. and Foster, J. A. (1998), "Removal bias: a new cause of code growth in tree based evolutionary programming," in *1998 IEEE International Conference on Evolutionary Computation*.

Soule, T., Foster, J. A., and Dickinson, J. (1996), "Code growth in genetic programming," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), pp 215–223, Stanford University, CA, USA: MIT Press.

Tackett, W. A. (1994), *Recombination, Selection and the Genetic Construction of Computer Programs*, PhD thesis, University of Southern California.

Vose, M. D. and Liepins, G. (1991), "Punctuated equilibria in genetic search," *Complex Systems*, (5):31–44.

Whigham, P. A. (1995), "A schema theorem for context-free grammars," in *1995 IEEE Conference on Evolutionary Computation*, volume 1, pp 178–181, Perth, Australia: IEEE Press.

Zhang, B.-T. and Muhlenbein, H. (1995), "Balancing accuracy and parsimony in genetic programming," *Evolutionary Computation*, 3(1):17–38.