Peter Nordin, Frank Francone and Wolfgang Banzhaf

In Genetic Programming, introns play at least two substantial roles: (1) A structural protection role, allowing the population to preserve highly-fit building blocks; and (2) A global protection role, enabling an individual to protect itself almost entirely against the destructive effect of crossover. We introduce Explicitly Defined Introns into Genetic Programming. Our results suggest that the introduction of Explicitly Defined Introns can improve fitness, generalization, and CPU time. Further, Explicitly Defined Introns partially replace the role of Implicit Introns (that is, introns that emerge from crossover and mutation without being explicitly defined as such). Finally, Explicitly Defined Introns and Implicit Introns appear, in some situations, to work in tandem to produce better training, fitness and generalization than occurs without Explicitly Defined Introns.

6.1 Introduction

Biological introns are portions of the genotype (the DNA) that are not expressed in the phenotype (the organism), [Watson et al. 1987]. In some eucaryotic cells, up to 70% of the genetic information is snipped out of the genome chemically before the creation of amino acids. Researchers have suggested that biological introns play some role in providing genetically safe areas for mutation and crossover, [Watson et al. 1987].

The operators in Genetic Programming – mutation and crossover – were designed by analogy to biology [Koza 1992]. Furthermore, like DNA, GP is variable in length. One might, therefore, expect introns to evolve in GP.

But GP introns should look different than biological introns. In most GP implementations, the genotype and the phenotype are one and the same. Unlike the biological model, non-essential parts of the GP genome cannot be "snipped out". In GP, therefore, the analog to biological introns would be evolved code fragments that do not effect the fitness of the individual. For example: $y = y + 0$.

The evolution of such code fragments has been repeatedly observed by GP researchers in tree based GP–some refer to the phenomenon as "bloating", [Tackett 1994], [Angeline 1994]. "Bloating" is the accumulation of apparently useless code in a GP population–that is, code that does not effect the fitness of the individual. Our research here confirms existence of the "bloating" phenomenon in linear GP structures, see Figure 6.1.

In this Chapter, we argue that introns appear in GP populations because they have an important function in evolution. As a result, the algorithm selects for the existence of GP introns in a wide variety of conditions. Introns are, however, a decidedly mixed blessing in GP. Because GP introns are stored in individual's evolved program structures instead of in a separate genome, a large amount of CPU time is spent calculating intron values.

In a recent paper, we began investigating introns. We devised a way to measure the intron content of genetically evolved programs using the linear structure of Compiling GP System (CGPS), [Nordin 1994]. In this paper, we continue that research by introducing Explicitly

Defined Introns (EDIs) into CGPS. An EDI is a structure in the CGPS system that plays no part in the fitness calculation but that affects the probability of crossover between adjacent blocks of evolved code–much like the biological intron. It does, however, affect the probability of crossover between the two Nodes on either side of the EDI (see "Definitions" and "Genetic Operators", below). By way of contrast, we will refer to introns that emerge from the code itself as "Implicit Introns" (IIs). The situation is depicted in Figure 6.2. The circular nodes affect the fitness calculation while the squares affect the crossover points. For instance the square with (14) inside acts as a string of 14 nodes during crossover but does not interfere with the fitness evaluation.

Previously, researchers have studied structures similar to our EDIs in fixed length GA representations, [Levenick 1991], [Forrest and Mitchell 1992]. This chapter is, apparently, the first application of EDIs to variable length EA structures or to Genetic Programming in particular.

Our results suggest that EDIs have the following effects in variable length representations:

1. Fitness, Generalization and CPU time frequently improve with the introduction of EDIs.

2. IIs and EDIs frequently work together, with IIs probably serving to chain EDIs together.

3. Under some circumstances, EDIs replace IIs in the population almost completely.

4. Like IIs, EDIs can, and frequently do, protect an entire individual or code block against the destructive effects of crossover

5. A combination of parsimony pressure and EDIs allow a population to keep the structural advantages of IIs without carrying some of the computational overhead of IIs.

6.2 Definitions

We have defined EDIs and IIs above. The following additional terms are needed to clarify the following discussion:

Node: The atomic crossover unit in the GP structure. Crossover can occur on either or both sides of a Node but not within a Node. Because our particular implementation of GP works with 32 bit machine code instructions (see below), a Node is a 32 bit instruction. A Node can be comprised of either Working Code (see definition below) or an II. An EDI is not a Node because it plays no role in the fitness calculation and because crossover occurs, effectively, within the EDI, not on either side of the EDI (see "Genetic Operators", below).

Working Code or Exon: A GP Node that is not an II. Working Code effects the fitness calculation of the individual for at least one fitness case.

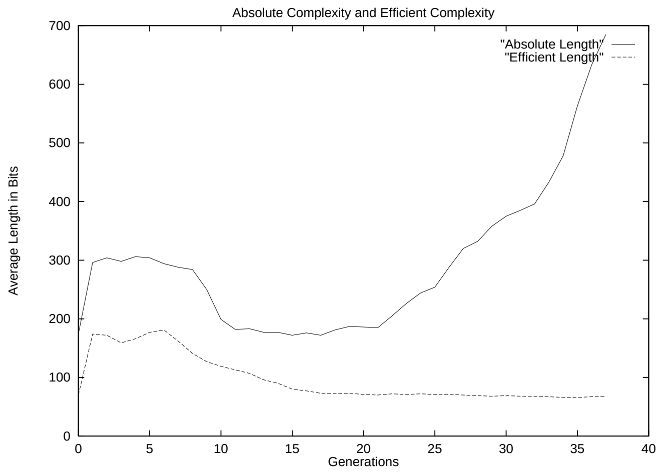

Figure 6.1
Growth of genome size during evolution, absolute and effective size. (Reproduced from [Nordin and Banzhaf 1995a].)

Absolute Size: The number of Nodes in a GP individual.

Effective Size: The number of Nodes in a GP individual that constitute Working Code–that is the number of Nodes in a GP individual that make a difference in the result of the individual's fitness calculation for at least one of the fitness cases. Figure 6.1 shows the evolution of effective and absolute size during training.

Intron Equivalent Unit: (IEU). An II or an EDI with a probability of crossover that is equal to an Implicit Intron comprised of a single Node. We will designate an II with an IEU value of 1 as *II*. We will designate an EDI with an IEU value of 1 as EDI(1). The purpose of defining this unit is to allow us to deal with both IIs and EDIs in one designation that consistently reflects their effect on the probability of crossover.

Explicitly Defined Intron Value: Each EDI stores an integer value which is initialized randomly through three different ranges. That Explicitly Defined Intron integer value shall be referred to as an "EDIV." This value affects the probability of crossover at the EDI, as discussed below in "Genetic Operators".

See Figure 6.2 for an illustration of EDIs, IIs and working code. Section 6.5.2.1 gives further details on the behavior of EDIs and IIs during evolution.

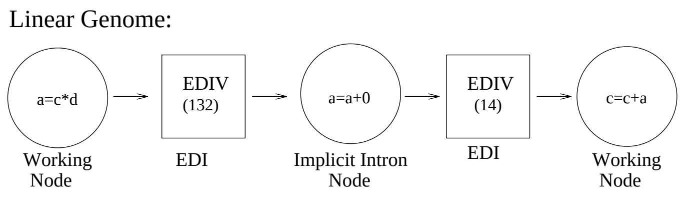

Figure 6.2
Explicitly Defined Introns, Implicit Introns, and Working Nodes in a linear Genome. (For the relation with tree structure based GP, see section 6.6)

6.3 The Experimental Setup

6.3.1 The Problem

We chose a straightforward problem of symbolic regression on a second order polynomial. Large constants for the polynomial and small terminal set ranges were deliberately chosen to prevent trivial solutions.

6.3.2 Runs

We chose 10 fitness cases and tested the best individuals for generalization on 10 data elements that were not included in the training set. Each run was performed on a population of 3000 individuals. We completed 10 runs each with and without parsimony (values: 0, 1), with and without EDIs enabled, and over three ranges of initialization for EDI values (values: high, medium, low). The total number of runs was 200 comprised of 240,000 individuals. Some additional runs were performed to investigate specific issues and will be described below.

6.3.3 Implementation of GP For This Problem

The Evolutionary Algorithm we use in this paper is an advanced version of the CGPS described in [Nordin 1994], composed of variable length strings of 32 bit instructions for a register machine. The register machine performs arithmetic operations on a small set of registers. Each instruction can also include a small integer constant of maximum 13 bits. The 32 bits in the instruction thus represents simple arithmetic operations such as "a=b+c" or "c=b*5". The actual format of the 32 bits corresponds to the machine code format of a SUN-4 [SPARC 1991], which enables the genetic operators directly to manipulate binary code. For a more thorough description of the system and its implementation see [Nordin and Banzhaf 1995b].

This implementation of GP makes it easier to define and measure intron sizes in code for register machines than in, for instance, functional S-expressions (see below). The setup is also motivated by fast execution, low memory requirement and a linear genome which makes reasoning about information content less complex.

6.3.4 Intron Measurements

Many classes of code segments with varying degree of intron behavior can be identified [Nordin and Banzhaf 1995a]. For instance:

1. Code segments where crossover never changes the behavior of the program individual for any input from the *problem domain*.

2. Code segments where crossover never changes the behavior of the program individual for any of the *fitness cases*.

3. Code segments which cannot contribute to the fitness and where each node can be replaced by a NoOperation without affecting the output for any input in the *problem domain*.

4. Code segments which do not contribute to the fitness and where each node can be replaced by a NoOperation without affecting the output for *any of the fitness cases*.

5. More continuously defined intron behavior where nodes are given a numerical value of their sensitivity to crossover.

The introns that we measure in this paper are of the fourth type.

We determine whether a Node is an II by replacing the Node with a NoOperation instruction. A NoOperation instruction is a neutral instruction that does not change the state of the register machine or its registers. If that replacement does not affect the fitness calculation of the individual for any of the fitness cases, the Node is classified as an Implicit Intron[1].

When this procedure is completed the number of first order introns is summed together as the intron length of that individual. Effective length is computed as absolute length less the intron length. The intron checking facility is computationally expensive but it operates in linear time in relation to the size of individuals.

[1] Note that this technique measures the presence of only first order introns. Examples of such intron segments with length one, called first order introns, are "a=a+0","b=b*1" etc. Higher order introns can also appear, such as the second order "a=a-1;a=a+1". In this case, the intron segment only acts as an intron as long as the two instructions are kept together. We chose to limit our measurement in this manner because observations and theoretical argumentation support the claim that higher order introns are a small proportion of the total intron length [Nordin and Banzhaf 1995a].

6.3.5 Genetic Operators

This section gives a brief description of the evolutionary operators used, For more details on the operators and the system, see [Nordin and Banzhaf 1995b].

Selection: fitness proportionate.

Crossover: Two arbitrary subsegments of Nodes are selected from a copy of each parent and then swapped to form the two children. If the two chosen segments are of different length, the length of the children will vary.

Crossover with EDIs: In runs using EDIs, the crossover point is selected just as if there were a chain of N nodes instead of the EDI with EDI-value ’N’. The crossover point is thus selected by examining the integer values (the EDIV) stored in the EDIs between Nodes in an individual. The probability of crossover between two Nodes is proportional to the EDIV of the EDI separating the Nodes. The EDIV values from two parents (k and n) are transmitted to the children as follows. EDIV(k) and EDIV(n) are summed. Then the sum is divided randomly between the EDIs that appear at the crossover point in the two children. This crossover operator performs *equivalent to crossing over two individuals in the middle of two chains of IIs*. We felt it was important to duplicate this phenomenon because of the frequency with which we have observed long chains of IIs in our prior work. In other words, crossover acts just as if every EDI was substituted by a string of normal introns with a length defined by the EDIV of the EDI. The values transmitted to the children corresponds to the values that would have been transmitted during a crossover with a normal intron (II) segments of this length.

Mutation: changes bits inside the 32 bits of the instruction (Node), which can change the operator, the source and destination registers, and the value of any constants.

The EDIV in each EDI is initialized as a uniform random distribution between a minimum and maximum value.

6.3.6 Parsimony Pressure

We used external parsimony pressure in some of our experiments. This feature of the system punishes Absolute Size in an individual by adding a parsimony factor times the Absolute Size of the individual to the fitness expression. A parsimony factor of *one* means that the Absolute Size of the individual is added to the computed fitness. Parsimony was never applied so as to penalize Explicitly Defined Introns and they could thus grow unaffected by the parsimony pressure.

Table 6.1 summarizes the parameters used during training in the 200 different training runs that constitute the basis for our analysis.

Table 6.1
Summary of parameters used during training.

Parameter name:	
Objective :	Symbolic regression of a polynomial with large constants
Terminal set :	Integers in the range 0-10
Function set :	Addition Subtraction Multiplication
Raw and stand. fitness :	The sum taken over the 10 fitness cases, of the absolute value of the difference between actual and desired value
Wrapper :	None
Maximum population size :	300, 3000
Crossover Prob :	90%
Mutation Prob :	5%
Selection :	Fitness proportional selection
Termination criteria :	Maximum number of Generations exceeded
Maximum number of generations:	150,1500
Parsimony Pressure :	0, 1, 5
EIDV init value :	10-20, 10-100, 10-1000
Maximum number of nodes:	512
Total number of experiments :	200

6.4 Protection Against Destructive Crossover

6.4.1 Definitions

The following terms have the following meanings:

Destructive Crossover A crossover operation that results in fitness for the offspring that is less than the fitness of the parents[2].

Constructive Crossover A crossover operation that results in fitness for the offspring that is more than the fitness of the parents.

Neutral Crossover A crossover operation that results in a combined fitness for the offspring that is within 2.5% of the fitness of the parents.

Figure 6.3 is a histogram that demonstrates the relative proportions of these three different types of crossover in a typical early generation in a typical run.

The x-axis gives the change in fitness $\Delta f_{percent}$ after crossover f_{after}.
($f_{best} = 0$, $f_{worst} = \infty$).

$$\Delta f_{percent} = \frac{f_{before} - f_{after}}{f_{before}} \cdot 100 \tag{6.1}$$

The area over zero represents Neutral Crossover, the area to the left of Zero represents Destructive Crossover and the area to the right of zero represents Constructive Crossover.

[2] At least 2.5% less fitness than the parents.

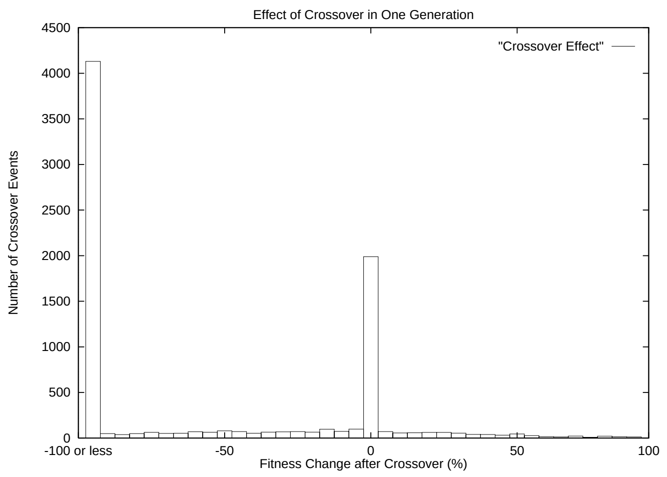

Figure 6.3
Typical Proportion of Destructive, Neutral and Constructive Crossover in an early generation. (Reproduced from [Nordin and Banzhaf 1995a].)

A three-dimensional extension of this figure is an important analysis tool for finding out what takes place during evolution. Figure 6.4 is constructed by compiling together, one figure of the same type as Figure 6.3 for each generation, thus enabling the study of distribution of crossover effect during a complete training session. In the example we see how the destructive crossover at the left decreases as the neutral crossover, in the middle, increases. The constructive crossover is some magnitudes lower in this figure and is therefore barely visible.

6.4.2 Effective Fitness and Protection Against Destructive Crossover

Using the concept of destructive crossover, we can formulate an equation describing the proliferation of individuals from one generation to the next, c.f. the Schema Theorem [Holland 1975]. For more details, see also [Nordin and Banzhaf 1995a].

Let C_{ej} be the effective size of program j, and C_{aj} its absolute size. Let p_c be the standard genetic programming parameter giving the probability of crossover at the individual level. The probability that a crossover in a segment of *Working Code* of program j will lead to a worse fitness for the individual is the probability of destructive crossover, p_{dj}. Let f_j be the fitness of the individual and $\overline{f^t}$ be the average fitness of the population in the current generation. If we use fitness proportionate selection and block exchange crossover, then for any program j the average proportion P_j^{t+1} of this program in the next generation is:

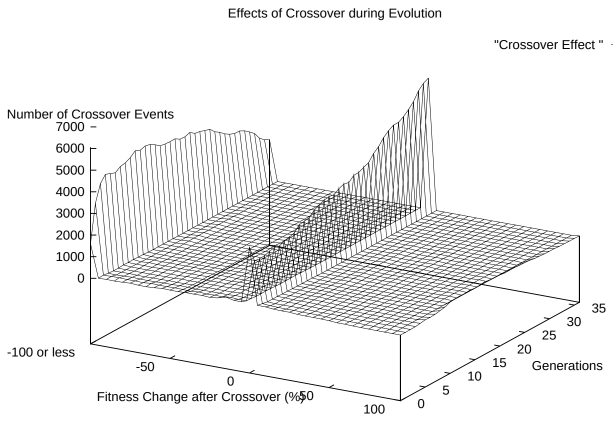

Figure 6.4
Distribution of Crossover Effects During Training. (Reproduced from [Nordin and Banzhaf 1995a].)

$$P_j^{t+1} \approx P_j^t \cdot \frac{f_j}{\overline{\overline{f^t}}} \cdot \left(1 - p_c \cdot \frac{C_{ej}}{C_{aj}} \cdot p_{dj}\right) \tag{6.2}$$

In short, equation 6.2 states that the proportion of copies of a program in the next generation is the proportion produced by the selection operator less the proportion of programs destroyed by crossover. We can interpret the crossover related term as a direct subtraction from the fitness in an expression for reproduction through selection. In other words, reproduction by selection and crossover acts as reproduction *by selection only*, if the fitness is adjusted by a term:

$$-p_c \cdot f_j \cdot C_{ej} \cdot \frac{1}{C_{aj}} \cdot p_{dj} \tag{6.3}$$

This could thus be interpreted as if there were a term equation 6.3 in our fitness proportional to program size.

We now define "effective fitness" f_{ej} as:

$$f_{ej} = f_j - p_c \cdot f_j \cdot C_{ej} \cdot \frac{1}{C_{aj}} \cdot p_{dj} \tag{6.4}$$

It is the *effective fitness* that determines the number of individuals of a certain kind in the next generation.

These equations suggest that there may be a number of strategies for an individual to increase its survival rate and the proportion of the next generations that contain its effective offspring. For example, it can:

1. Improve its fitness

2. Increase its absolute size

3. Decrease its effective size

4. Rearrange its code segments to be less vulnerable to crossover.

5. Take advantage of special representation to reduce the probability of destructive crossover.

In this paper, we address the later four strategies for an individual to improve its effective fitness in the following terms:

Global protection of an individual against the destructive effects of crossover, refers to changes in effective fitness caused by changes in Absolute or Effective Size.

Structural protection refers to changes in effective fitness caused by rearranging the proportion of IIs among code fragments to better protect the high fitness code from crossover.

Finally, EDIs provide a special representation that, we argue, can increase effective fitness by allowing the individual to improve both its *global* protection and its *structural* protection.

6.4.3 Intron Protection Analysis

Introns thus can protect against Destructive Crossover in at least two different ways: Global protection is protection of all the Working Code of an individual against destructive crossover, while structural protection is protection of portions of an individual's Working Code against destructive crossover. Either can improve the *effective fitness* of an individual.

6.4.3.1 Structural Protection Against Crossover

Let A, B, and C represent Nodes of Working Code. Let *II* represent a Node that is an Implicit Intron. Consider the following two individuals:

$$A - B - II - C, with\ 3\ possible\ crossover\ points. \tag{6.5}$$

$$A - B - C, with\ 2\ possible\ crossover\ points. \tag{6.6}$$

The probability that crossover will occur at the A-B block of code in equation 6.5 is 33%. In equation 6.6, it is 50%. Therefore, in equation 6.5, the A - B block of code has greater structural protection against destructive crossover than the same block of code in equation 6.6. If the A - B block of code is highly fit, then the individual in equation 6.5 has a higher "effective fitness" than the individual in equation 6.6.

6.4.3.2 Global Protection Against Crossover

IIs can also protect an entire individual from the destructive effects of crossover. Consider the following two individuals:

$$A - B - C - II \tag{6.7}$$

$$A - B - C \tag{6.8}$$

The probability that crossover will occur in a manner that will disrupt the Working Code (A, B, C) in equation 6.7 is 66%. In equation 6.8, the probability is 100%. Therefore, the Working Code in equation 6.7 is better protected against destructive crossover than it is in equation 6.8 and the individual in equation 6.7 has higher "effective fitness" than the individual in equation 6.8.

6.4.3.3 EDIs and Protection Against Crossover

The probability of crossover between the Nodes separated by an EDI is proportional to the EDIV of that EDI (see "Genetic Operators", infra).

In the following individual, therefore, the A - B code block is relatively more protected against being disrupted by crossover than the B - C block.

$$A - EDI(1) - B - EDI(2) - C \tag{6.9}$$

Likewise, the first of the following individuals is more highly protected against any of its Working Code (A and B) being disrupted by crossover than the second individual.

$$A - EDI(1) - B - EDI(100) - II \tag{6.10}$$

$$A - EDI(1) - B - II \tag{6.11}$$

6.4.3.4 Intron Equivalent Units (IEUs) and Protection Against Crossover

We have defined the IEU value between any two Nodes as the sum of the number of II Nodes and the sum of the EDIVs between these two Nodes. Because our analysis suggests that both IIs and EDIs should provide both global and structural protection against destructive crossover, we predict that the IEU value between two Nodes should be a good measure of the amount of both global and structural crossover protection between those Nodes.

We also predict that runs that use the EDI component of IEUs instead of the II component may save CPU time because the fitness calculation of an individual is unaffected by EDIs. IIs, of course, consume as much CPU time as the Working Code. Similarly the CPU time consumed by crossover can be reduced by the less expensive computation and selection of number of Nodes.

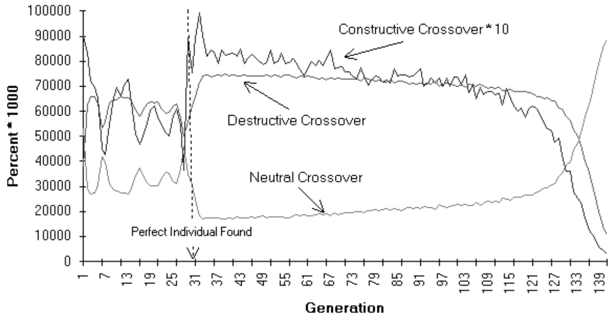

Figure 6.5
Destructive, Neutral and Constructive Crossover by Generation. Constructive Crossover Is Multiplied By 10 For Scaling Purposes. EDI enabled. Parsimony = 0.

6.5 Experimental Results

We divide this discussion into three sections. The first addresses the global effect of Intron Equivalent Units on protecting the entire individual from the destructive effects of crossover. The second addresses the structural effect of IEUs on protecting blocks of code from the destructive effects of crossover. The third discusses the effects of EDIs on fitness, generalization and CPU time.

6.5.1 Global Protection Against Crossover

Throughout most generations of all runs that we have measured, Destructive Crossover is by far the most prevalent effect of the crossover operation. (See Figures 6.3 and 6.4, infra). However, toward the end of most runs (with one notable exception discussed below) Destructive Crossover falls rapidly to but a fraction of its initial state. Figure 6.5 shows the relative amounts of the different types of crossover by generation for a typical run.

In Figure 6.5, Constructive Crossover appears much higher than its actual level because, for scaling purposes, it is multiplied by 10.

At about generation 125, the proportion of Destructive Crossover events in Figure 6.5 starts to fall rapidly. By generation 141, it is only 10% of the total incidents of crossover. Obviously, something is protecting the population from Destructive Crossover. The protection comes, we believe, from the concurrent growth of IEUs. Figure 6.6 shows the growth of IEUs for the same run. At about generation 125, both IIs, absolute size and EDIV increase rapidly. The IEU is simply the number of IIs plus the total EDIV. Thus, the

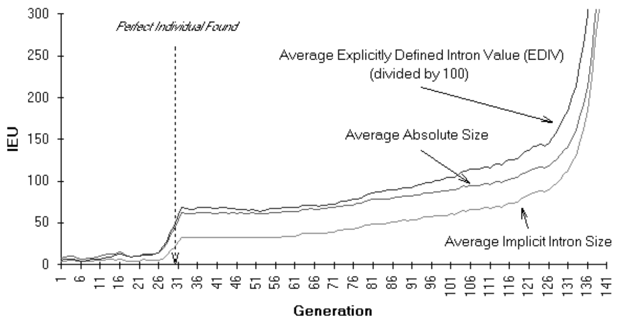

Figure 6.6
Average Absolute Size, Average Implicit Intron Size and Average EDIV Value by Generation. EDI enabled. Parsimony = 0. For scaling purposes, EDIV is divided by 100.

Table 6.2
Percent of Runs Where Destructive Crossover Disappeared. (Zero Destructive Crossover Events per Generation.) In the last line the Destructive Crossover did not fall but stabilized at around 70%.

EDI enabled	Parsimony	No. Runs	Population	Percent
Yes	0	10	300 each	100%
Yes	1	10	300 each	100%
No	0	10	300 each	100%
No	1	10	300 each	0%

predicted global protection effect of IEUs on destructive crossover appears validated.

In these examples we continue measurements after the best individual is found. Note, however, that there is no way for the system to “know” that this has happened. The system dynamics are the same as if the system gets stuck in a local optima for a shorter or longer period. The dynamics of rapid intron growth is typical for a system were fitness is hard to improve. The same phenomenon can be observed when the system is temporarily stuck at a local optima.

To test the persistence of this phenomenon, we performed the following forty runs for up to 1500 generations with the middle range of EDIV initialization, see table 6.2. In thirty out of the forty runs described in Table 6.2, Destructive Crossover eventually fell to less than 10% of the total Crossover Events. In those runs that we have examined, the fall in Destructive Crossover was always accompanied by a comparable increase in the Average IEU value for the population.

The ten runs in which Destructive Crossover never fell, were runs in which there were no EDIs and there was a parsimony factor. See Table 6.2. These runs are the exceptions that prove the rule. In these runs, the parsimony measure forces the number of IIs to almost zero early in training. There were, of course, no EDIs in these runs. As a result, the entire population has a low IEU value throughout the run. Destructive Crossover, therefore, remains very high.

On the other hand, when EDIs are enabled, Destructive Crossover does fall below 10% in every run – even where there is a parsimony penalty. In these runs, IIs never grow rapidly – instead the EDIVs undergo the rapid growth late in training. Thus, the total Intron Equivalent Units (IIs plus EDIVs) undergo rapid growth late in training in these runs. In so doing, they apparently inhibit Destructive Crossover in the same way that IIs do when there is no parsimony factor, see Table 6.2. In short, whenever IEUs grew rapidly, Destructive Crossover fell at the same time. Whenever IEUs did not grow rapidly, Destructive Crossover did not fall.

Table 6.2 suggests that the correlation between rapid IEU growth and an equally rapid decline in destructive crossover is 1.0. This fact confirms our prediction about the effect of IEUs on global protection against crossover.

6.5.2 Structural Protection Against Crossover

We argued above that IEUs, which include both IIs and EDIs, have the theoretical ability to protect blocks of code from the destructive effects of crossover. We predict that the evolutionary algorithm may use this capability to protect blocks of code that bear the highest likelihood of producing fitter offspring. In Altenberg's terminology we predict that IEUs may increase survivability of blocks of code with high constructional fitness and, therefore, increase the evolvability of the population as a whole, [Altenberg 1994].

6.5.2.1 Constructional Fitness and Protection Against Crossover

Assume that the code block, A - B, is Working Code and has a relatively high constructional fitness. Assume also that the code block K - A is also Working Code but has a relatively low constructional fitness. Consider the following two individuals with the following IEU configurations: $m = n \; and \; k > j$:

$$Parent1 : K - EIU(n) - A - EIU(j) - B; \tag{6.12}$$

$$Parent2 : K - EIU(m) - A - EIU(k) - B. \tag{6.13}$$

The offspring of Parent 1 are more likely to survive than the offspring of Parent 2 because they are more likely to contain the more fit block, A - B. Thus, we would expect the A - B code block from Parent 1 to multiply through the population more rapidly than the A - B code block from Parent 2.

At first blush, this would imply that the Average IEU per individual in a population should *decrease* as training continues. After all, the A - B code block from the first configuration will take its low EIU value with it when it replicates unbroken. However, there is another factor at work.

Consider two similar individuals but with $m > n \; and \; j = k$:

$$Parent1 : K - EIU(n) - A - EIU(j) - B \tag{6.14}$$

$$Parent2 : K - EIU(m) - A - EIU(k) - B. \tag{6.15}$$

In this configuration, Parent 2's offspring are more likely to survive because the highly fit block, A - B is less likely to be disrupted by crossover. As m increases, so does the amount of protection accorded to the A - B block. This factor would tend to increase the IEU in low fitness blocks of code as training continues.

Two countervailing factors should, therefore, be at work in setting the Average IEU Value for a population during the early, constructional phases of training–pressure to decrease the IEU values in blocks with high constructional fitness and pressure to increase the IEU values in blocks with low constructional fitness. However, the lowest possible value of j and k is 1. On the other hand, there is no upper limit to the value of m or n. Accordingly, selection pressure should work by providing pressure to increase the values of m and n.

We do not, however, expect the balance of these two factors to result in an exponential increase in overall IEU values during the early and constructional phase of training [Altenberg 1994]. There is yet a third factor at work. While the exponential increase strategy works well for highly fit individuals at the end of training (Figure 6.6), it would be counterproductive when constructive crossover is still likely early in training. Consider Figures 6.5 and 6.6. The result of rapidly increasing IEU values is a dramatic increase in neutral crossover. In the early stages of training, individuals that have such high IEU values that they are protected globally from crossover will not survive long. Their contemporaries, who still allow for constructive crossover, will eventually surpass their performance.

One other factor must be considered. There is one significant difference in the way IIs and EDIs function. Where parsimony does not altogether suppress the emergence of IIs, IIs are capable of chaining EDIs together. However, EDIs are not capable of chaining together other EDIs. We expect, therefore, to find differences in the behavior of EDIs and IIs depending on the parsimony factor. It is also possible that we will find EDIs and IIs working together in chains, instead of entirely supplanting each other.

Thus, we expect that the amount of IEU in a population will be a result of the balance of the above three factors plus the ability of IIs to string together EDIs. This theory suggests that, on balance, the evolutionary algorithm should select for the presence of IIs and IEUs during the constructional phase of training, but not exponentially. A finding that the the evolutionary algorithm was not selecting for or against the amount of IEU in the population or that it was selecting against the presence of IEU would be inconsistent with our hypothesis

6.5.2.2 Testing Intron Equivalent Unit Contribution To Constructional Fitness

Although the global protection effect, discussed above, is dramatic, it really only signals that the training is decisively over. Spotting the effect is rather easy as long as there is a way to measure Intron Equivalent Units. Measuring the structural effect of IIs and EDIs is considerably more difficult than measuring the global effect. Unlike the global protection effect, we would not expect the structural protection effect to cause easily measurable exponential increases in EDIVs or the number of IIs. We were, nevertheless, able to devise two tests for our hypothesis.

Test 1. Measuring Selection for IEU Values.
We discarded the absolute level of IEUs per individual in the population as a good measure of whether or not the evolutionary algorithm is or is not selecting for the presence of IEUs because of the "hitchhiking" phenomenon. Researchers have pointed out that one way useless code replicates throughout the population is by hitchhiking with adjacent blocks of highly fit code [Tackett 1994]. Our findings are not in any way inconsistent with this observation. But the hitchhiking phenomenon implies that Average IEU per individual would be a poor way to measure whether the evolutionary algorithm is selecting for the presence of IEUs. Because the average amount of Working Code changes substantially as training progresses, we would expect the amount of hitchhiking IEUs in the population also to fluctuate. Thus, the hitchhiking phenomenon precludes Average IEU per individual as a good measure of whether the evolutionary algorithm is selecting for or against the presence of IEUs.

Instead, we chose Average IEUs per Node in the population as our measure. In other words, we look at the average of the sum of the IIs and the EDIVs per Node. This measure eliminates the possibility that we are really measuring changes in IEUs caused by hitchhiking instead of measuring whether or not the evolutionary algorithm is selecting for or against the presence of IEUs. Here are the predictions we make regarding this measure:
• If our hypothesis is false and hitchhiking is the only source for IEU growth, then IEU per Node should remain more or less constant or fluctuate randomly until the late stages of training.
• If our hypothesis is correct, on the other hand, IEU per node should increase during early training, but not exponentially.

We calculated Average IEU per Node over 80 runs with and without parsimony and with and without EDIs. We then plotted that figure, along with Average Best Individual Fitness, by generation. The results are reported below.

The tests using no EDIs and a parsimony measure were not helpful in evaluating our hypothesis one way or the other. There were no EDIs to measure and the parsimony measure suppressed the growth of IIs. Since these are the only two components of IEUs, we regard any result from these runs as unhelpful either way.

The other three tests were considerably more helpful. Figures 6.7, 6.8, and 6.9 show the results of these three tests over 60 runs with 180,000 individuals in their populations.

The results with all three parameter sets are consistent with our hypothesis. Figures 6.7 and 6.8 illustrate this with the greatest clarity. Average IEU per Node increases during training until the Average Best Individual Fitness over the 10 runs stops improving. At

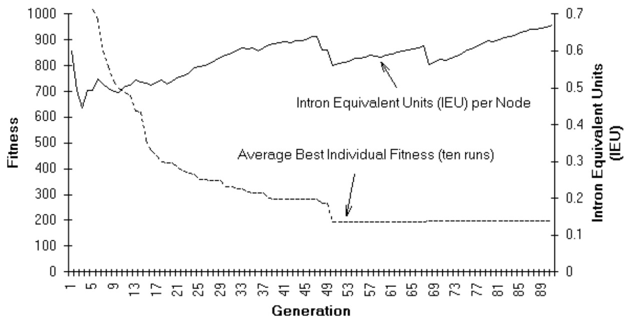

Figure 6.7
IEU per Node and Best Individual Fitness By Generation Averaged over Ten Runs with No EDI and No Parsimony

Table 6.3
EDI parameters

EDI enabled ?	Parsimony	No Runs	Population
Yes	0	10	3000
Yes	1	10	3000
No	0	10	3000

that point IEU per Node drops. After that IEU per Node again rises. A climb in IEU per Node until Best Individual Fitness stops improving is consistent with our prediction that the evolutionary algorithm will select for the existence of IEUs during the constructional phase of training.

Figure 6.9, which presents the results with EDIs enabled and no parsimony measure, also shows a clear pattern. The figure is built from data over 40 runs. The data in figure 6.9 is scaled or normalized. The normalization point is the generation where the best individual was found.

In summary, the results of Test 1 suggest that IIs and EDIs are not merely passive units during training. If they were, we would not expect to find evidence that the evolutionary algorithm was actively selecting for their presence. We conclude, therefore, that Test 1 is consistent with our hypothesis and inconsistent with the notion that IIs and EDIs are only to be regarded as useless code.

Test 2. Measuring Interactions between IIs and EDIs.

Our hypothesis also predicts that EDIs and IIs may interact with each other–either replacing each other or working together or both. If EDIs and IIs are merely useless code, IIs should

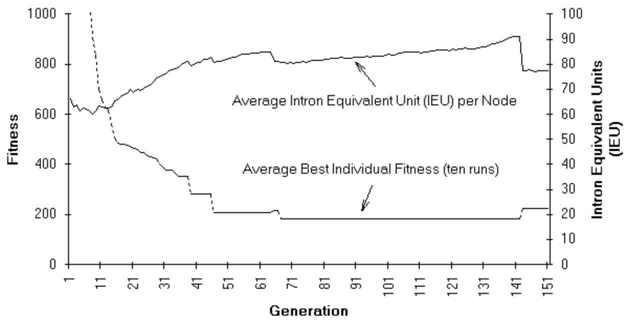

Figure 6.8
IEU per Node and Best Individual Fitness By Generation Averaged over Ten Runs with EDI Enabled and Parsimony = 1

Table 6.4
Effect of adding EDIs on the Percentage of the Average Absolute Size of Individuals that is Composed of IIs. For runs that found a perfect individual.

EDIs Enabled ?	Parsimony	Implicit Introns	Sample Size (No. of Individuals)
No	0	53%	9,000
Yes	0	45%	9,000
No	1	25%	21,000
Yes	1	19%	18,000

come and go of their own accord unaffected by the presence or lack EDIs in the population.

We tested for such interactions in two ways. First, we measured the percentage of average Absolute Size of the population that was comprised of Implicit Introns. This test was performed on the same runs used in Test 1. The test was performed with and without EDIs enabled. We then measured the effect of adding EDIs on the percentage of IIs[3]. Table 6.4 contains the results[4].

In runs that found a perfect individual, the addition of EDIs reduced the percentage of IIs in the population at the time a perfect individual was found to a significant degree.

When the same figures for all forty runs was examined, the same pattern was found, the percentage of IIs in the population drops when EDIs are added. However, the drop in all runs was less than half the drop for the runs that found a perfect individual. This suggests that the runs that did best (that is, runs that found a perfect individual), were the runs in

[3] Because we do not include EDIs in the measure of Absolute Size, the addition of the EDIs cannot effect this measurement except indirectly, by effecting the number of IIs.

[4] Because the point where a run finds the best individual appeared to be important in our prior reported results, we measured the change in percentage at the point in each run where best individual fitness stopped improving.

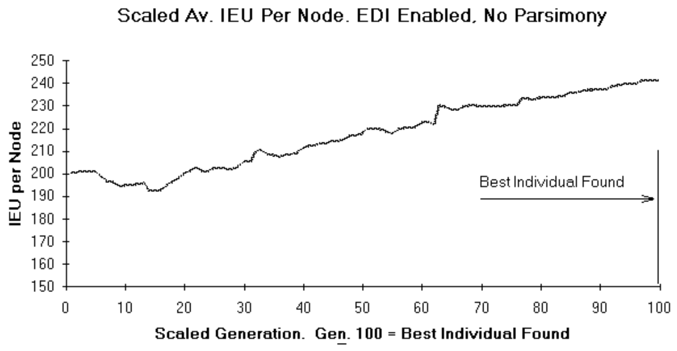

Figure 6.9
IEU per Node and Best Individual Fitness By Generation Averaged over Ten Runs with EDI Enabled and No Parsimony. Scaled and Normalized Around Generation of Best Individual.

which EDIs replaced IIs to the greatest extent.

However this data is viewed, it supports the notion that, to some extent, EDIs replace IIs when EDIs are added to the population and may do so more in runs that successfully find a perfect individual. This is consistent with our hypothesis and inconsistent with the notion that IIs and EDIs are merely "useless" code.

The second method we used to test for interactions was as follows. We looked for evidence that IIs and EDIs work together. One such interaction is very suggestive. Refer back to Figure 6.6. With no parsimony measure in this run, IIs and EIVs apparently change their values in lockstep. When one changes, so does the other. When we added a parsimony factor, the result was very different. Figure 6.10 details that run using the same random seed.

Note that before the discovery of a perfect individual, the EDIV and the II values in Figure 6.10 move in lockstep. What is important here is that, despite the parsimony pressure, IIs persisted in the population in a proportion more typical of a run without a parsimony factor. As soon as the best individual is found, the number of IIs drops to, effectively, zero – much more typical of our parsimony runs.

We interpret Figures 6.6 and 6.10 as evidence that the presence of EDIs can improve the survival value of IIs until a perfect individual is found. After a perfect individual is found, IIs lose that selection advantage and revert to their normal pattern in runs using parsimony. We speculate that this effect of adding EDIs is based on the ability of IIs to string together long chains of EDIs and thereby rapidly increase or decrease the amount of protection accorded to various blocks of code. EDIs, by themselves do not have this ability.

In any event, were EDIs and IIs merely blocks of useless code, we would not expect such

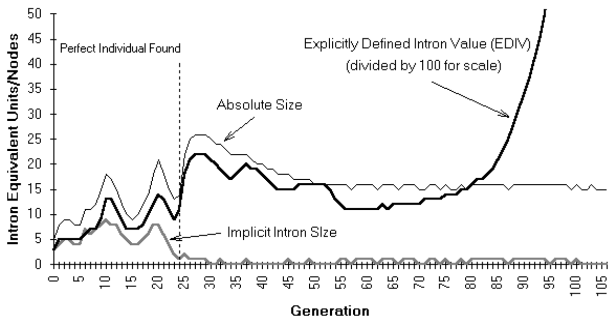

Figure 6.10
Explicitly Defined Intron Value and Implicit Intron Size for Typical Run With EDI Enabled and Parsimony = 1.

an interaction between them. This evidence, also, is consistent with our hypothesis.

One intriguing possibility raised by Figure 6.10 is that the average II values in a population may be a very practical way to improve training, at least where a parsimony factor is used. There are two possibilities.

First, we ran these same tests with a much higher parsimony measure–parsimony factor equal to 5. In those runs, the IIs were suppressed altogether and did not demonstrate the pattern in Figure 6.10. Fitness and generalization were both worse when the higher parsimony factor was used. It may be that the pattern in Figure 6.10 means that the parsimony factor is "just right." Looking for this pattern in preliminary runs may be a good way to set the parsimony factor.

Second, if the pattern in Figure 6.10 persists for other types of problems – that is, if II size generically fluctuates until the best individual is found and then falls to close to zero, this measure may be a way to determine when to stop training. Having a measure of when the population can do no better would be an invaluable tool in GP training. We regard this as an important area for further research.

Conclusion Regarding The Structural Role of IEUs

Both of the tests we devised tend to reject the hypothesis that, in the early stages of training, IIs and EDIs are only useless code that happens to be adjacent to highly fit blocks of code. Rather, the results suggest that IIs and EDIs can play an important role in finding the most highly fit individual.

Proving that EDIs and IIs are not useless do not, by itself, prove our position that the role played by IIs and EDIs is to protect code blocks with high constructional fitness from

Table 6.5
Average Fitness of Best Individual, All Runs (smaller is better)

EDI Enabled	Parsimony	EDI Range	Average Fitness
Yes	1	Medium	108
Yes	0	Narrow	122
No	1	N/A	156
Yes	1	Narrow	186
Yes	0	Wide	196
No	0	N/A	280
Yes	1	Wide	290
Yes	0	Medium	354

Table 6.6
Average Generalization of Best Individual, All Runs (smaller is better)

EDI Enabled	Parsimony	EDI Range	Average Generalization
Yes	0	Narrow	413
Yes	1	Medium	473
Yes	0	Wide	537
No	1	N/A	542
Yes	1	Narrow	632
Yes	1	Wide	815
No	0	N/A	860
Yes	0	Medium	1176

the destructive effects of crossover. It is merely consistent with such a role. Some of the evidence is highly suggestive of such a role and the theoretical reasoning that they can play such a structural role is strong. However, we regard this as an area ripe for further research.

6.5.3 Effect of EDIs on Fitness, Generalization and CPU Time

The addition of EDIs profoundly affect every measure of performance. In summary, the best performances in the various categories in rank order are set forth in Tables 6.5, 6.6, and 6.7: Table 6.5 shows the average fitness for 10 runs each of different combinations of parsimony, EDI, and initializations. The best result were achieved with EDIs, parsimony pressure and a medium sized initializations range of EDIVs, see table 6.1 for EDIV init values.

Table 6.6 illustrates the performance of the best individuals on unseen data. Each average is over 10 runs and shows best performance for a system with EDIs.

The last table shows the CPU time needed to reach a perfect individual. Unlike IIs, EDIs does not affect the fitness calculation and the system can therefore search faster. The average CPU time is lowest when EDIs are enabled.

By any measure, EDIs proved capable of improving the performance of the algorithm. EDIs were enabled for the best two categories for each measure of performance.

However, runs with EDIs seem quite sensitive to the range with which the EDIs were initialized. One example illustrates possible pitfalls of training with EDIs and directions

Table 6.7
Average Time in Seconds To Find Perfect Individual.

EDI Enabled	Parsimony	EDI Range	Average CPU Time
Yes	1	Wide	31
Yes	1	Narrow	56
No	1	N/A	56
Yes	1	Medium	59
Yes	0	Medium	98
Yes	0	Wide	116
No	0	N/A	164
Yes	0	Narrow	179

for further research. By far the best average CPU time to find a perfect individual was with parsimony equal to 1 and the "Wide" initialization range for EDIs. Yet the average fitness in that same category was toward the bottom of the list. The reason is that only four of the ten runs in this category found a perfect individual. Those four runs, however, found the perfect individuals very quickly.

The reason for this apparent discrepancy may be that protection against destructive crossover is a two edged sword. While the wide range of EDIs helped these runs to find a perfect individual very quickly, it may also have helped the remaining 6 runs find local minima quickly–and get stuck there. A little less protection against crossover (the narrow and medium ranges for the same parameters) resulted in slower CPU performance but 60% of the runs found perfect individuals.

6.6 Future Work

We would like to extend our current results to a more canonical GP system with hierarchical crossover and tree representation [Koza 1992]. That would also shed light on any potential differences in behavior during evolution induced by representation and crossover operators. We have so far done a few initial experiments with a canonical GP system doing symbolic regression. The results indicate a distribution of destructive crossover similar to that in the system used in this paper. Figure 6.11 shows the distribution of crossover effect of S-expression based GP system doing symbolic regression over 60 generations. Figure 6.11 suggests that our results may apply to a wider domain of systems, see also [Rosca 1995]. In that regard, we plan to perform intron size measurements in a tree based GP system.

In addition, we believe that further investigation into the structural effects of IEUs is warranted, as well as investigations into continuously defined intron properties, see section 6.3.4. We would also like to study how exons and introns are distributed in the genome during evolution.

Finally, we believe that EDIs must be tested on real world problems with more intractable solution spaces.

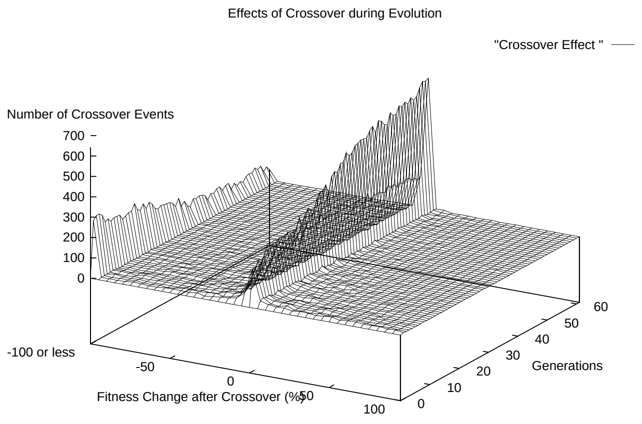

Figure 6.11
Crossover Effects In S-Expression Style GP

Acknowledgments

We would like to thank Walter Tackett for the use of his Genetic Programming system, from the GP-archive. This research has been supported by the Ministry for Wissenschaft und Forschung (MWF) of Nordrhein-Westfalen, under grant I-A-4-6037.I .

Biblography

Altenberg, L. (1994) The Evolution of Evolvability in Genetic Programming. In *Advances in Genetic Programming*, K. Kinnear, Jr. (ed.), Cambridge, MA: MIT Press. pp47-74.

Angeline P.J. (1994) Genetic Programming and Emergent Intelligence In *Advances in Genetic Programming*, K. Kinnear, Jr. (ed.), Cambridge, MA: MIT Press.

Forrest, S. and M. Mitchell (1992) Relative building block fitness and the building block hypothesis In *Foundations of Genetic Algorithms 2*, D. Whitley (ed.). San Mateo, CA: Morgan Kaufmann Publishers Inc., pp 109-126.

Holland, J. (1975) *Adaption in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press.

Koza, J. (1992) *Genetic Programming*, Cambridge, MA: MIT Press.

Levenick, J.R. (1991), Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue From Biology. In *Proceedings of the Fourth International Conference on Genetic Algorithms*,Belew, R.K. and Booker, L.B., editors, Morgan Kauffman. San Mateo, California, pp. 123-7.

Nordin J.P. (1994) A Compiling Genetic Programming System that Directly Manipulates the Machine-Code. In *Advances in Genetic Programming*, K. Kinnear, Jr. (ed.), Cambridge, MA: MIT Press.

Nordin J.P, Banzhaf W. (1995a) Complexity Compression and Evolution. *In proceedings of Sixth International Conference of Genetic Algortihms,* Morgan Kaufmann Publishers Inc.

Nordin J.P, Banzhaf W. (1995b) Evolving Turing Complete Programs for a Register Machine with Self-Modifying Code. *In proceedings of Sixth International Conference of Genetic Algortihms,* Morgan Kaufmann Publishers Inc.

Rosca J.P. (1995) Entropy-Driven Adaptive Representation. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* Technical Report 95.2 June 1995 University of Rochester, USA

The SPARC Architecture Manual,(1991), SPARC International Inc., Menlo Prak, California.

Tackett, W.A. (1995). Greedy Recombination and Genetic Search on the Space of Computer Programs. In *Foundations of Genetic Algorithms III*, Whitley, D. and Vose, M. Eds.. Morgan Kaufmann, San Mateo, California.

Watson J.D, Hopkins N.H, Roberts J.W, Wiener A.M, (1987) *Molecular Biology of the Gene*, Menlo Park, CA: The Benjamin/Cummings Publishing Company, Inc.