SysReport 3/95
System Analysis
Department of Computer Science
University of Dortmund, Germany

Explicitly Defined Introns and Destructive Crossover in Genetic Programming

Peter Nordin* Universität Dortmund
Frank Francone† FRISEC
Wolfgang Banzhaf‡ Universität Dortmund

Abstract

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.

1 Introduction

Introns are an important part of the genomes of eucaryotic cells. In some genes, up to 70 % of the DNA sequence is not expressed in amino acids. Watson et al, therefore, classify this material as introns–genetic code that does not apparently express itself in the individual produced by the genome [1].

Genetic Programming (GP) makes heavy use of methods borrowed from natural evolution [2]. Genetic Programming systems work with variable length code. This factor permits the evolution of code that does not effect the fitness of the individual. In GP, therefore, the analog to naturally evolved introns would be the evolution of code fragments that do not effect the fitness of the individual.

In fact, such code fragments do appear in evolved GP populations. Tackett has pointed to the tendency of GP structures to undergo "bloat" [4], Figure 2. Simply put, "bloat" is a term used to describe the accumulation in a GP population of apparently useless code (that is, code that does not effect the result calculated by a GP individual). Researchers, borrowing a term from biology, have referred to these fragments of apparently "useless" code as "introns."

In a recent paper, we begun investigating introns by devising a way to measure the first order intron content of genetically evolved programs using the linear genomes of a Compiling GP system (CGPS) [3]. Other researchers have studied introns already earlier in other fixed-length, evolutionary algorithms [5, 6]. We continue our research into introns here.

In summary, we argue that introns in GP are not just "useless" code. Rather, they play at least two significant roles in training. Specifically, we introduce "Explicitly Defined Introns" (EDI's) into GP structures. An EDI is an instruction segment that is inserted between two Nodes. It can, by definition, only act as an intron. It can *not* effect the calculated result of the individual. It does, however, effect the probability of crossover between the two Nodes on either

*Universität Dortmund, Fachbereich Informatik , Lehrstühl fur Systemanalyse , D–44221 Dortmund, email: nordin@ls11.informatik.uni-dortmund.de
†email: ytns65a@prodigy.com
‡email: banzhaf@ls11.informatik.uni-dortmund.de

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" (II's). The situation is depicted in Figure 1.

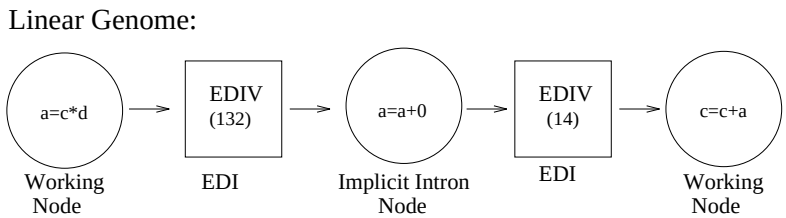

Figure 1: Explicitly Defined Introns, Implicit Introns, and Working Nodes in a linear Genome

Previously, researchers have experimented with EDI's in fixed length GA structures with some suggestion that EDI's can enhance preservation of better building blocks. [5, 6] Our results suggest that EDI's have the following effects in variable length GP structures:

1. Fitness, Generalization and CPU time frequently improve with the introduction of EDI's (Explicitly Defined Introns).
2. II's (Implicit Introns) and EDI's frequently work together, with II's probably serving to chain EDI's together.
3. Under some circumstances, EDI's replace II's in the population almost completely.
4. Like II's, EDI's can, and frequently do, protect an entire individual or code blocks against the destructive effects of crossover.
5. A combination of parsimony pressure and EDI's allows a population to keep the structural advantages of II's without carrying some of the computational overhead of II's.

2 Definitions

We have defined EDI's (Explicitly Defined Introns) and II's (Implicit Introns) above. The following additional terms will tend to clarify our 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 (Implicit Intron). An EDI (Explicitly Defined Intron) 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 (Implicit Intron). Working Code effects the fitness calculation of the individual for at least one fitness case.

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 2 shows the evolution of effective and absolute size during training.

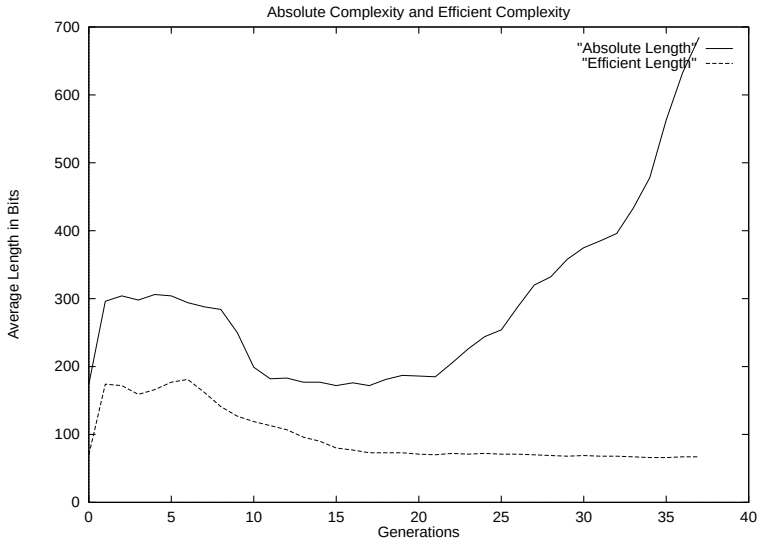

Figure 2: Growth of genome size during evolution, absolute and effective size. (Reproduced from [3].)

Intron Equivalent Unit (IEU). An II (Implicit Intron) or an EDI (Explicitly Defined Intron) 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 II's and EDI's in one designation that consistently reflects their effect on the probability of crossover.

Explicitly Defined Intron Value. Each EDI (Explicitly Defined Intron) 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".

3 The Experimental Setup

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.

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 EDI's enabled, and over three ranges of initialization for EDI values (values: high, medium, low). The total of runs was 80 comprised of 240,000 individuals. Some additional runs to look at specific issues were performed and will be described below.

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 [7], 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 [8], which enables the genetic operators directly to manipulate binary code. For a more thorough description of the system and its implementation see [9].

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.

3.4 Intron Measurements

Many classes of code segments with varying degree of intron behavior can be defined [3]. 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 (Implicit Intron) 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.

3.5 Genetic Operators

Below follows a brief description of the evolutionary operators used, For more details on the operators and the system, see [9].

Selection: fitness proportionate.

[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 [3].

Table :	
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
Parameters :	
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
EIV init value :	10-20, 10-100, 10-1000
Maximum number of nodes:	512
Total number of experiments :	200

Table 1: Summary of parameters used during training.

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 EDI's: In runs using EDI's (Explicitly Defined Introns), the crossover point is selected by examining the integer values (the EDIV) stored in the EDI's 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 EDI's that appear at the crossover point in the two children. This operator allows crossover equivalent to crossing over two individuals in the middle of two chains of II's. We felt it was important to duplicate this phenomenon because of the frequency with which we have observed long chains of II's in our prior work.

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.

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.

Table 1 summarizes the parameters used during training in the 200 different training runs that constitute the basis for our analysis.

4 Protection Against Destructive Crossover

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 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.

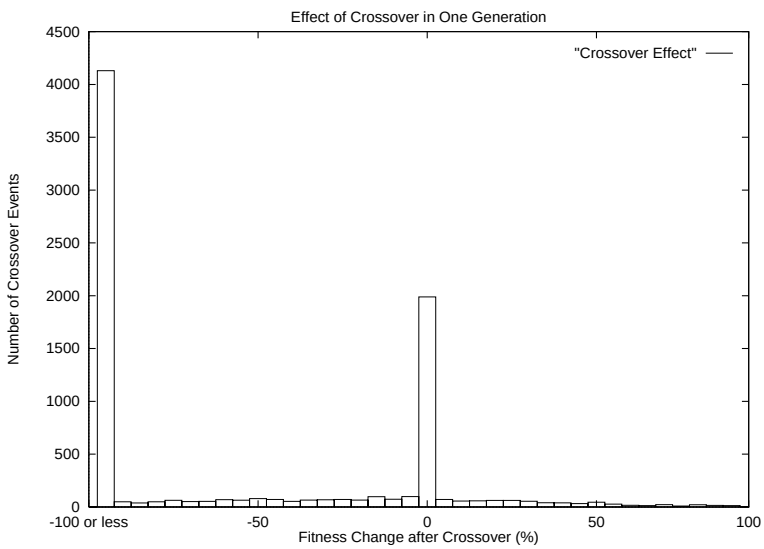

Figure 3: Typical Proportion of Destructive, Neutral and Constructive Crossover in an early generation. (Reproduced from [3].)

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

$$\Delta f_{percent} = \frac{f_{before} - f_{after}}{f_{before}} \cdot 100 \tag{1}$$

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.

A three-dimensional extension of this Figure is an important analysis tool for finding out what takes place during evolution. Figure 4 is constructed by compiling together, one figure of the same type as Figure 3 for each generation, thus enabling the study of distribution of crossover effect during a complete training session.

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

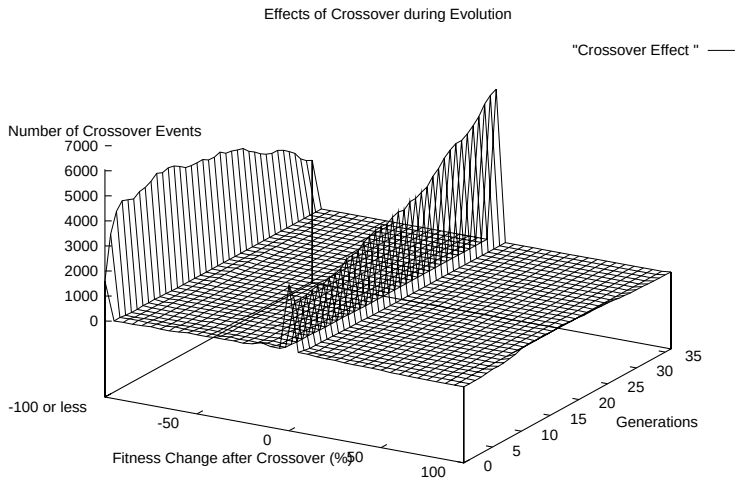

Figure 4: Distribution of Crossover Effects During Training. (Reproduced from [3].)

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. For more details, see [3].

Let C_{ej} be the effictive 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 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:

$$P_j^{t+1} \approx P_j^t \cdot \frac{f_j}{\overline{f^t}} \cdot \left(1 - p_c \cdot \frac{C_{ej}}{C_{aj}} \cdot p_{dj}\right) \tag{2}$$

In short, equation (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:

$$-p_c \cdot f_j \cdot C_{ej} \cdot \frac{1}{C_{aj}} \cdot p_{dj} \tag{3}$$

This could thus be interpreted as if there were a term (3) in our fitness proportional to program size.

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

$$f_{ej} = f_j - p_c \cdot f_j \cdot C_{ej} \cdot \frac{1}{C_{aj}} \cdot p_{dj} \tag{4}$$

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 rearraging the proportion of II's among code fragments to better protect the high fitness code from crossover.

Finally, EDI's 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.

4.3 Intron Protection Analysis

Introns can protect against Destructive Crossover in at least two different ways: structurally or globally. By global protection, we mean protection of all the Working Code of an individual against destructive crossover. By structural protection, we mean protection of portions of an individual's Working Code against destructive crossover. Either can improve the *effective fitness* of an individual.

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:

$$A - B - II - C, with\ 3\ possible\ crossover\ points. \tag{5}$$

$$A - B - C, with\ 2\ possible\ crossover\ points. \tag{6}$$

The probability that crossover will occur at the A-B block of code in (5) is 33%. In (6), it is 50%. Therefore, in (5), the A - B block of code has greater structural protection against destructive crossover than the same block of code in (6). If the A - B block of code is highly fit, then the individual in (5) has a higher "effective fitness" than the individual in (6).

4.3.2 Global Protection Against Crossover

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

$$A - B - C - II \tag{7}$$

$$A - B - C \tag{8}$$

The probability that crossover will occur in a manner that will disrupt the Working Code (A, B, C) in (7) is 66%. In (8), the probability is 100%. Therefore, the Working Code in (7) is better protected against destructive crossover than it is in (8) and the individual in (7) has higher "effective fitness" than the individual in (8).

4.3.3 EDI's and Protection Against Crossover

The probability of crossover between the Nodes separated by an EDI (Explicitly Defined Intron) is proportional to the Explicitly Defined Intron Value (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.

$$A - EDI(1) - B - EDI(2) - C \tag{9}$$

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.

$$A - EDI(1) - B - EDI(100) - II \tag{10}$$

$$A - EDI(1) - B - II \tag{11}$$

4.3.4 Intron Equivalent Units (IEU's) and Protection Against Crossover

We have defined the IEU value between any two Nodes as the sum of the number of Implicit Intron (II) Nodes and the sum of the Explicitly Defined Intron Values (EDIV's) beween these two Nodes. Because our analysis suggests that both II's and EDI's should provide both global and structural proctection 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 IEU's instead of the II component may save CPU time because the fitness calculation of an individual is unaffected by EDI's. II's, 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.

5 Experimental Results

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

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 3 and 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 5 shows the relative amounts of the different types of crossover by generation for a typical run.

In Figure 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 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 Intron Equivalent Units (IEU's). Figure 6 shows the growth of IEU's for the same run. At about generation 125, both Implicit Introns (II's), absolute size and EDIV (Explicitly Defined Intron Values) increase rapidly. The IEU is simply the number of II's plus the total EDIV. Thus, the predicted global protection effect of IEU's on destructive crossover appears to translate into reality.

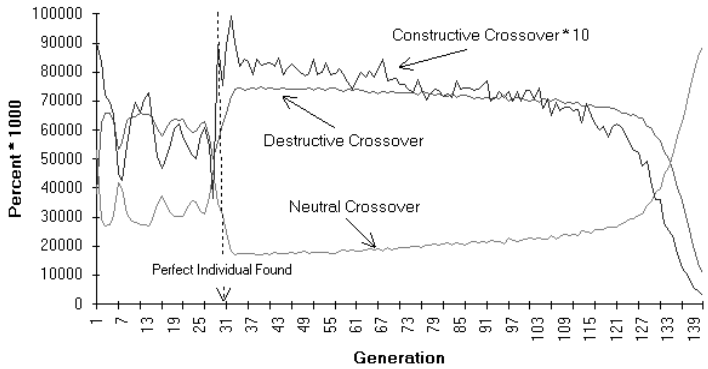

Figure 5: Destructive, Neutral and Constructive Crossover by Generation. EDI enabled. Parsimony = 0.

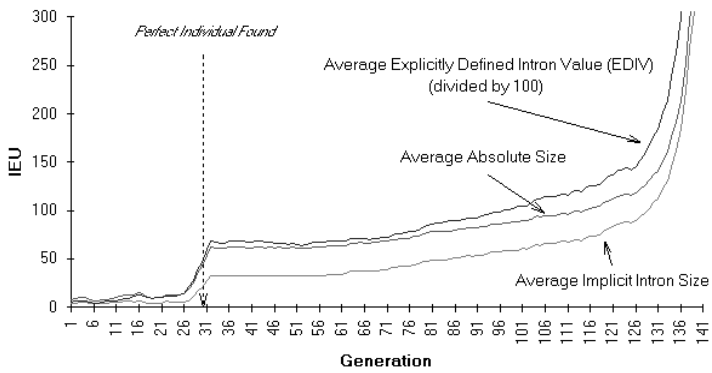

Figure 6: Average Absolute Size, Average Implicit Intron Size and Average EDIV Value by Generation. EDI enabled. Parsimony = 0. For scaling purposes, EDIV is divided by 100.

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 2. In thirty out of the forty runs described in Table 2, Destructive Crossover fell to less than 10% of the total Crossover Events, eventually. 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 EDI's and there was a parsimony factor. See Table 2. These runs are the exceptions that prove the rule. In these runs, the parsimony measure forces the number of II's (Implicit Introns) to almost zero early in training. There were, of course, no EDI's 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 EDI's (Explicitly Defined Introns) are enabled, Destructive Crossover does fall below 10% in every run–even where there is a parsimony penalty. In these runs, II's never grow rapidly–instead the EDIV's undergo the rapid growth late in training. Thus, the total Intron Equivalent Units (II's plus EDIV's) undergoes rapid growth late in training in

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%

Table 2: Percent of Runs Where Destructive Crossover Fell to 10% of the Total Incidents of Crossover

these runs. In so doing, they apparently inhibit Destructive Crossover in the same way that II's do by themselves when there is no parsimony factor See Table 2. In short, whenever IEU's grew rapidly, Destructive Crossover fell at the same time. Whenever IEU's did not grow rapidly, Destructive Crossover did not fall.

Table 2 suggests that the correlation between rapid IEU growth and an equally rapid decline in destructive crossover is 1.0. This fact confirm our prediction about the effect of IEU's on global protection against crossover.

5.2 Structural Protection Against Crossover

We argued above that IEU's (Intron Equivalent Units), which include both II's (Implicit Introns) and EDI's (Explicitly Defined Introns), 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 IEU's may increase survivability of blocks of code with high constructional fitness and, therefore, increase the evolvability of the population as a whole, [10].

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 Intron Equivalent Unit (IEU) configurations: $m = n \ and \ k > j$:

$$Parent1 : K - EIU(n) - A - EIU(j) - B; \tag{12}$$

$$Parent2 : K - EIU(m) - A - EIU(k) - B. \tag{13}$$

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 highly 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 $m > n \ and \ j = k$:

$$Parent1 : K - EIU(n) - A - EIU(j) - B \tag{14}$$

$$Parent2 : K - EIU(m) - A - EIU(k) - B. \tag{15}$$

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 putting upward pressure on 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 [10]. 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), it would be counterproductive when constructive crossover is still likely early in training. Consider Figures 5 and 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 long survive. Their contemporaries, who still allow for constructive crossover, will pass them by.

One other factor must be considered. There is one significant difference in the way II's (Implicit Introns) and EDI's (Explicitly Defined Introns) function. Where parsimony does not altogether suppress the emergence of II's, II's are capable of chaining EDI's together. However, EDI's are not capable of chaining together other EDI's. We expect, therefore, to find differences in the behavior of EDI's and II's depending on the parsimony factor. It is also possible that we will find EDI's and II's 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 II's to string together EDI's. This theory suggests that, on balance, the evolutionary algorithm should select for the presence of II's and IEU's 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

5.2.2 Testing Intron Equivalent Unit (IEU) 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 II's and EDI's 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 EDIV's or the number of II's. We were, nevertheless, able to devise two tests for our hypothesis.

Test 1. Measuring Selection for IEU Values.

We discarded the absolute level of IEU's 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 IEU's because of the "hitchhiking" phenomonon. Researchers have pointed out that one way useless code replicates throughout the population is by hitchhiking with adjacent blocks of highly fit code [4]. Our findings are not in any way inconsistent with this observation. But the hitchhiking phenomenon implies that Average IEU (Intron Equivalent Unit) per individual would be a poor way to measure whether the evolutionary algorithm is selecting for the presence of IEU's. Because the average amount of Working Code changes substantially as training progresses, we would expect the amount of hitchhiking IEU's 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 IEU's.

Instead, we chose Average Intron Equivalent Units (IEU's) per Node in the population as our measure. In other words, we look at the average of the sum of the II's and the EDIV's per Node. This measure eliminates the possibility that we are really measuring changes in IEU's caused by

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

Table 3: EDI parameters

hitchhiking instead of measuring whether or not the evolutionary algorithm is selecting for or against the presence of IEU's. Here are the predictions we make regarding this measure:

First: 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.

Second. If our hypothesis is correct, on the other hand, IEU per node should increase during early trainnng, but not exponentially.

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

The tests using no EDI's and a parsimony measure were unhelpful in evaluating our hypothesis one way or the other. There were no EDI's to measure and the parsimony measure suppressed the growth of II's. Since these are the only two components of IEU's, we regard any result from these runs as unhelpful either way.

The other three tests were considerably more helpful. Figures 7, 9, and 8 show the results of these three tests over 60 runs with 180,000 individuals in their populations.

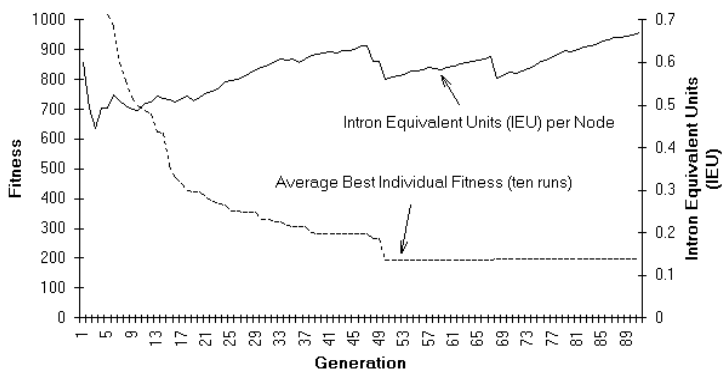

Figure 7: IEU per Node and Best Individual Fitness By Generation Averaged over Ten Runs with No EDI and No Parsimony

The results with all three parameter sets are consistent with our hypothesis. Figures 7 and 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 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 IEU's during the constructional phase of training.

Figure 9, which presents the results with EDI's enabled and no parsimony measure, is somewhat more ambiguous. Although IEU per Node also rises until the discovery of a best individual, there are periods of apparently exponential increase. When we examined the cause of these exponential increases, we discovered that several of the runs in the average quickly found a best individual and then went through exponential growth while other runs had still not found a

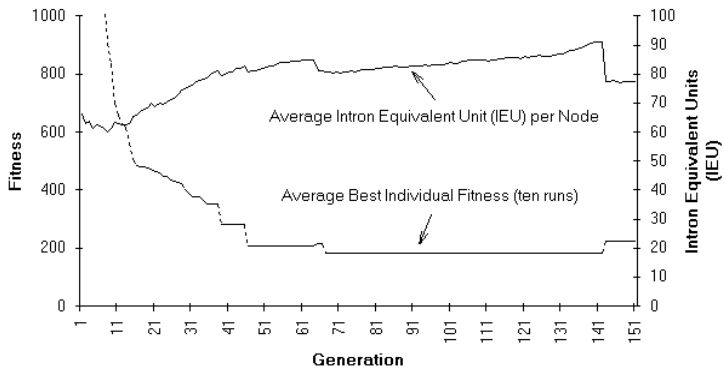

Figure 8: IEU per Node and Best Individual Fitness By Generation Averaged over Ten Runs with EDI Enabled and Parsimony = 1

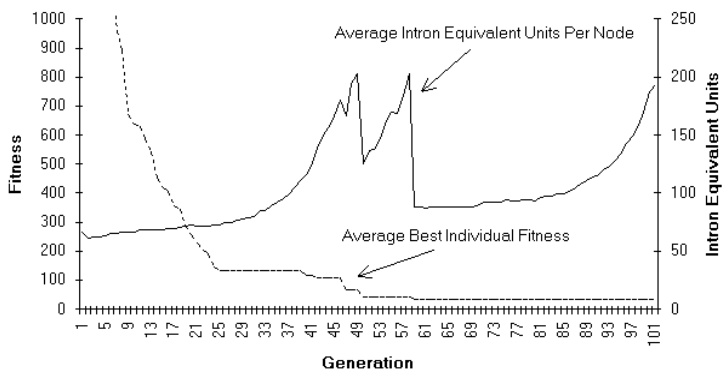

Figure 9: IEU per Node and Best Individual Fitness By Generation Averaged over Ten Runs with EDI Enabled and No Parsimony

best individual. This accounted for the exponential increases before all 10 runs found a best individual. We note that before the exponential increases begin, the pattern in Figure 9 is very similar to the pattern in Figures 7 and 8. We regard Figure 9 as, therefore, not inconsistent with our hypothesis.

In summary, the results of Test 1 suggest that II's and EDI's 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 II's and EDI's are only useless code.

Test 2. Measuring Interactions between II's and EDI's.

Our hypothesis also predicts that EDI's (Explicitly Defined Introns) and II's (Implicit Introns) may interact with each other–either replacing each other or working together or both. If EDI's and II's are merely useless code, II's should come and go of their own accord unaffected by the presence or lack EDI's 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

EDI's 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

Table 4: Effect of adding EDI's on the Percentage of the Average Absolute Size of Individuals that is Composed of II's. For runs that found a perfect individual.

comprised of Implicit Introns. This test was performed on the same runs used in Test 1. The test was performed with and without EDI's enabled. We then measured the effect of adding EDI's on the percentage of II's[3]. Table 4 contains the results[4].

In runs that found a perfect individual, the addition of EDI's reduced the percentage of II's 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 II's in the population drops when EDI's 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 which EDI's replaced II's to the greatest extent.

However these data are viewed, they support the notion that, to some extent, EDI's replace II's when EDI's 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 II's and EDI's are merely "useless" code.

Second: We looked for evidence that II's and EDI's work together. One such interaction is very suggestive. Refer back to Figure 6. With no parsimony measure in this run, II's and EIV's 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 10 details that run using the same random seed.

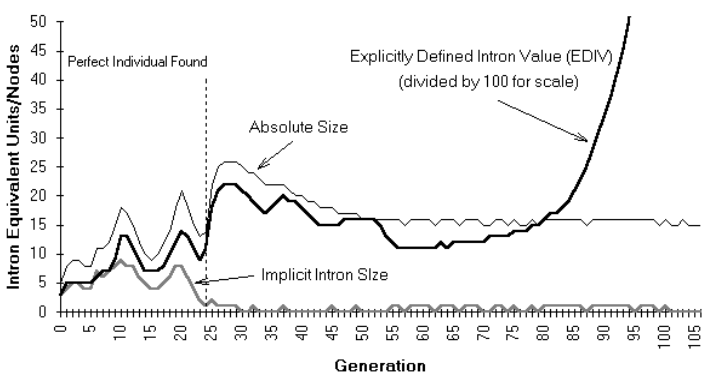

Figure 10: Explicitly Defined Intron Value (EDIV) and Implicit Intron (II) Size for Typical Run With EDI Enabled and Parsimony = 1.

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

[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.

Note that before the discovery of a perfect individual, the EDIV and the II values in Figure 10 move in lockstep. What is important here is that, despite the parsimony pressure, II's 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 II's drops to, effectively, zero–much more typical of our parsimony runs.

We interpret Figures 6 and 10 as evidence that the presence of EDI's can improve the survival value of II's until a perfect individual is found. After a perfect individual is found, II's lose that selection advantage and revert to their normal pattern in runs using parsimony. We speculate that this effect of adding EDI's is based on the ability of II's to string together long chains of EDI's and thereby rapidly increase or decrease the amount of protection accorded to various blocks of code. EDI's, by themselves do not have this ability.

In any event, were EDI's and II's merely blocks of useless code, we would not expect such an interaction between them. This evidence, also, is consistent with our hypothesis.

One intriguing possibility raised by Figure 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 = 5. In those runs, the II's were suppressed altogether and did not demonstrate the pattern in Figure 10. Fitness and generalization were both worse when the higher parsimony factor was used. It may be that the pattern in Figure 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 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 IEU's

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

One final note. In the next section, we show how EDI's frequently improved fitness, generalization and CPU time. Were EDI's merely useless code, one would not expect such an effect.

Proving that EDI's and II's are not useless does not, by itself, prove our position that the role played by II's and EDI's is to protect code blocks with high constructional fitness from 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.

5.3 Effect of EDI's on Fitness, Generalization and CPU Time

The addition of EDI's profoundly affect every measure of performance. In summary, the best performances in the various categories in rank order are set forth in Tables 5, 6, and 7:

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

However, runs with EDI's are seem quite sensitive to the range with which the EDI's were initialized. One example illustrates possible pitfalls of training with EDI's and directions for further research. By far the best average CPU time to find a perfect individual was with parsimony = 1 and the "Wide" initialization range for EDI's. 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

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 5: Average Fitness 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

Table 6: Average Generalization of Best Individual, All Runs (smaller is better)

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

Table 7: Average Time in Seconds To Find Perfect Individual. (For Runs that Found A Perfect Individual)

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 EDI's 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 Future Work

We would like to extend our current results to a more canonical GP system with hierarchical crossover and tree representation [2]. 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 11 shows the distribution of crossover effect of S-expression based GP system doing symbolic regression over 60 generations. Figure 11 suggests that our results may apply to a wider domain of systems.

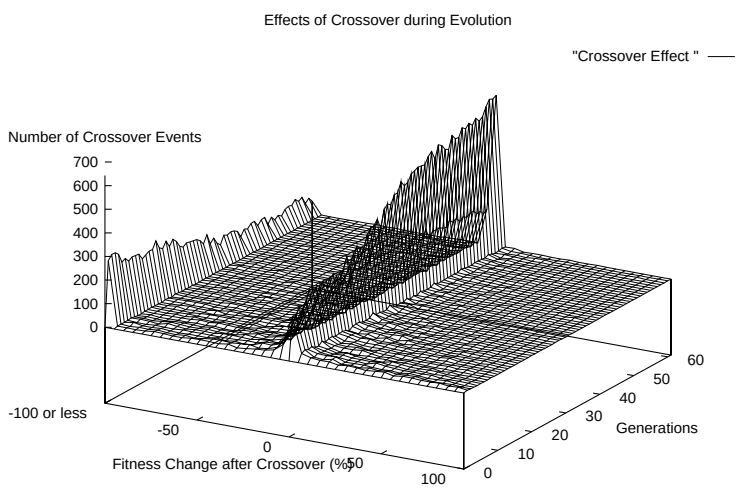

Figure 11: Crossover Effects In S-Expression Style GP

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 IEU's is warranted, as well as investigations into continuously defined intron properties, see section 3.4.

We would also like to study how exons and introns are distributed in the genome during evolution.

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

7 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 .

References

[1] 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.

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

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

[4] 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.

[5] 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.

[6] 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.

[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.

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

[9] Nordin J.P, Banzhaf W. (1995) 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.

[10] 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.