Duplication of Coding Segments in Genetic Programming

Thomas Haynes
The University of Tulsa
600 South College Avenue
Tulsa, OK 74104–3189
e–mail: haynes@euler.mcs.utulsa.edu

Abstract

Research into the utility of non–coding segments, or introns, in genetic–based encodings has shown that they expedite the evolution of solutions in domains by protecting building blocks against destructive crossover. We consider a genetic programming system where non–coding segments can be removed, and the resultant chromosomes returned into the population. This parsimonious repair leads to premature convergence, since as we remove the naturally occurring non–coding segments, we strip away their protective backup feature. We then duplicate the coding segments in the repaired chromosomes, and place the modified chromosomes into the population. The duplication method significantly improves the learning rate in the domain we have considered. We also show that this method can be applied to other domains.

Introduction

In genetic–based encodings (GBE), a *bit* is the atomic element of a chromosome and a *non–coding segment* is a continuous collection of bits (≥ 1) that do not contribute to the overall fitness of a chromosome. Non–coding segments in chromosomes are actively being investigated in both genetic algorithms (GA) and genetic programming (GP) (Nordin 1996; Wu & Lindsay 1995). They facilitate the evolution of solutions in domains by guarding against destructive crossover by providing bits where the exchange of genetic material will not effect the fitness of the chromosome. Duplication of coding segments is found in GP chromosomes, and is believed to be the building blocks for GP (Tackett 1993). Multiple appearances of a building block increases the probability that it will survive reproduction. GP research has a difficulty in identifying building blocks for a domain (O'Reilly 1995; Rosca 1995). We implement a domain in which all building blocks can be enumerated, allowing us to investigate the effects of duplication of building blocks within chromosomes.

We present an approach in which all non–coding segments are removed from a GP chromosome. A non–coding segment, which is a duplicate of the coding segment, is spliced onto the chromosome. At least one of the resultant children is guaranteed to be at least as fit as the parent. We then examine the utility of increasing the number of copies of the coding segment within the chromosome.

Non–coding segments

Non–coding segments model the intragenic regions reported in the biological literature and are the intron segments seen in the GBE literature. They account for a large fraction of the DNA (Futuyma 1986) and are believed to be backup material for the coding segments. For example, the frog *Xenopus laevis* has 450 copies of the gene codings for 18S and 28S rRNA and 24,000 copies of the gene for 5S rRNA (Futuyma 1986). The non–coding sequences might also act as a library for adaptation. During RNA splicing the non–coding sequences are stripped, producing a smaller RNA molecule. As the gene can be spliced in a variety of ways, the non–coding sequence for one mRNA could be a coding sequence for another (Alberts *et al.* 1989). As a protein evolves to meet changes in the environment, it can also resort to the non coding segments instead of evolving entirely new genetic material.

Genetic Algorithms

In the GBE literature, the emphasis on non–coding segments has focused on how these extra bits provide a buffer against destructive crossover. The canonical GA chromosome, or string, representation utilizes a binary alphabet. If a `don't care` symbol is utilized, we have the schemata alphabet $\{0, 1, *\}$. A schemata is a template describing subsets of strings within the string. The *defining length* of a schema is the distance between the outermost bits defined on the binary alphabet. *Building blocks* have a small defining length and are highly fit. They are integral to the schema

theorem, which defines how the implicit parallel search of a GA "builds" better solutions over time.

The addition of non–coding segments to chromosomes separates building blocks and protects them from being sliced by crossover (Levenick 1991). GA chromosomes are typically of fixed length. With a string of length l, and a building block of defining length δ, any crossover operation has a probability

$$P_l = \frac{\delta}{l-1}$$

of destroying a building block (Goldberg 1989). If non–coding segments, of a total length of i are added, then the probability of destructive crossover breaking up a building block of defining length δ decreases to

$$P_{l+i} = \frac{\delta}{l+i-1}.$$

An example of non–coding segments is shown in Figure 1(a): there is a string of length $l = 15$ and a building block, $b1$, of defining length $\delta = 6$. The probability of crossover destroying $b1$ is $P_l = 0.43$. In Figure 1(b) a non-coding segment of length $i = 5$ is added and the probability of destructive crossover decreases to $P_{l+i} = 0.32$. Adding the non–coding segment to the chromosome's tail reduces the probability of destructive crossover, but does not aid the recombination of building blocks as much as placing the non–coding segments between the building blocks (Wu & Lindsay 1995).

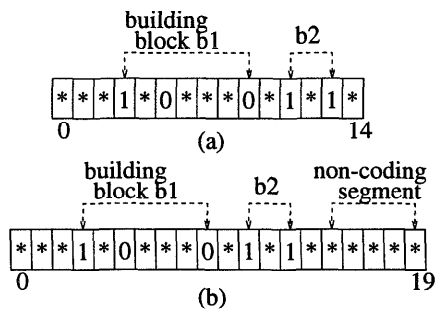

Figure 1: Non–coding segments in GA chromosomes prevent destructive crossover. (a) Without the non–coding segment. (b) With the non–coding segment.

The key to inserting non–coding segments into the GA chromosome is that they reduce the chance of destructive crossover. An "artificial" constraint is that only the coding segments are examined to determine the fitness of a chromosome. Therefore material within a non–coding segment cannot be mixed with that within the coding segment. Thus the non–coding segment material is meaningless, and selection pressure does not drive it to be backup material.

Wu and Lindsay (Wu & Lindsay 1995) point out that there is a drawback to inserting non–coding segments: they retard the growth of building blocks. It is hard for evolution to recombine the building blocks if non–coding segments are there to prevent destructive crossover. However, once those building blocks are formed, they are quite difficult to break up.

Genetic Programming

The "basic" theory of GP is borrowed from that of GA. Due to the difficulties in detecting building blocks in GP chromosomes, research is ongoing into formally connecting the theory as to why GP works with that of why GAs work (O'Reilly 1995). The canonical GP chromosome representation is a parse tree (S–expression). The difference between GA and GP is more than the fixed versus variable genotype representation. In GA there is a close relationship between the genotype and phenotype structure of a chromosome. Thus the building blocks of GAs are usually represented at the genotype level, and building blocks are relatively easy to detect. With the GP, building blocks are at the phenotype or semantical level, and are difficult to represent, detect, and capture. There can also be a duplication of building blocks in a GP chromosome, whereas there may not be any such duplication in a GA chromosome.

Tackett (Tackett 1993) compares the difficulty in researching building blocks between GP and GA: different notations of schemata and a non–binary alphabet. He believes that small subtrees which appear frequently in S–expressions are GP's building blocks. These subtrees are prevalent due to their contribution to the fitness of the chromosomes in which they appear.

Altenberg (Altenberg 1994) believes duplications appear inside GP chromosomes due to two selection forces adding blocks of code to the population. The genetic operators spread a block to different chromosomes, and an emergent selection pressure causes the formation of duplication within a chromosome. The duplication is a result of the fitness of the block being replicated. Angeline (Angeline 1994) reports while there is redundancy in chromosomes, the benefit of these semantically extraneous components is in the prevention of destructive crossover. He highlights a difference between GAs and GPs with regards to non–coding segments: in GAs they are added by design and in GPs they evolve naturally.

Nordin (Nordin 1996) investigates the dynamics of non–coding segments in GP evolution. His chromosomes are comprised of linear genomes which are 32 bit strings and are binary code for a SUN–4. Non–coding bits are defined to be those that when replaced

by a NOP instruction do not change the semantics or phenotype of the chromosome. Using this capability, Nordin investigated the effects of non–coding segments on destructive crossover. He reached the same conclusions regarding the utility of non–coding segments as did the GA researchers. He reports promising preliminary results with the canonical representation.

Clique Detection

Given a graph $G = (V, E)$, a clique of G is a complete subgraph of G, and is denoted by the set of vertices in the complete subgraph. The goal is to find all cliques of G. Since the subgraph of G induced by any subset of the vertices of a complete subgraph of G is also complete, it is sufficient to find all maximal complete subgraphs of G (Kalmanson 1986). A maximal complete subgraph of G is referred to as a maximal clique. Figure 2 is a 10 node graph, with cliques:

$$C = \{ \{0,3,4\}, \{0,1,4\}, \{1,4,5\}, \{1,2,5\}, \{2,5,6\}, \{3,4,7\}, \{4,7,8\}, \{4,5,8\}, \{5,8,9\}, \{5,6,9\}\}.$$

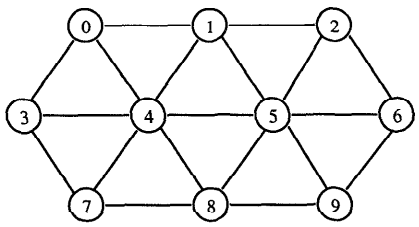

Figure 2: Example 10 node graph.

A variable length chromosome is needed because in general there will be an unknown number of cliques per graph. Potential cliques are denoted as *candidate cliques*. The chromosome is then a collection of candidate cliques. Candidate cliques are tested to determine if they: contain duplicate nodes, are subsumed by another candidate clique, are completely connected, and are maximal.

Each chromosome in the population represents sets of candidate cliques. The function and terminal sets are $F = \{ExtCon, IntCon\}$ and $T = \{1,\ldots,\#nodes\}$. *ExtCon* connects two sets of candidate cliques, while *IntCon* connects nodes inside a candidate clique. The fitness evaluation has rewards for both clique size and number of cliques. To detect the maximal connected subgraphs, the reward for size must be greater than that for the number of cliques. The algorithm to evaluate the chromosome is:

1. Parse it into a ordered series of candidate cliques.
2. Discard invalid candidates, i.e., repeated nodes.
3. Discard candidates subsumed by other candidates.
4. Discard candidates not completely connected.

If α and β are constants which are configurable by the user, $c = \#$ *of valid candidate maximal cliques*, and $n_i = \#$ *nodes in clique*$_i$, then the formula for measuring the fitness is:

$$F = \alpha c + \sum_{i=1}^{c} \beta^{n_i}.$$

Approach

We utilize a strongly typed GP (STGP) (Montana 1995) system instead of a canonical GP system to force type inheritance. A serious constraint on the user–defined terminals and functions in GP systems is *closure*, i.e. all of the functions must accept arguments of a single data type and return values of that type. STGP allows for an additional level of typing to be added. We have extended STGP by adding type inheritance to allow for more than two levels of typing (Haynes, Schoenefeld, & Wainwright 1996). In the context of the clique detection domain, we are forcing the chromosome to evolve "lists" of nodes.

The fitness function for the clique detector pares the chromosome down to the coding segments. The list of candidate cliques for a given chromosome succinctly encapsulates the content of that chromosome. Each candidate clique is a building block from which "better" chromosomes can be constructed. This paring down of the chromosome is similar to the RNA splicing in that non–coding segments are stripped out of the RNA transcript from DNA.

If a GA chromosome has invalid bits, and an algorithm can translate those bits into valid bits, then they can be repaired and the resultant chromosome evaluated to determine the fitness of the original chromosome. Issues are whether or not to return the repaired chromosome into the population and at what rate of return (Orvosh & Davis 1993). Repair is done at chromosome evaluation, not during the reproduction stage; there is no assurance that the repaired chromosome will even be selected for reproduction.

The evaluation function maps chromosomes from GP space to clique set space, i.e. genotype to phenotype. Repair maps the phenotype back into a genotype. Since the evaluation function removes nodes that do not contribute to the fitness, the resultant chromosome is likely to be smaller than the original. An example chromosome for the 10 node graph is presented in Figure 3. It has five candidate cliques, and the only cliques are #2 and #5: $C = \{\{4,8,7\}, \{5,6\}\}$. The others are eliminated because they violate at least one of the rules: #4 contains duplicate nodes, i.e. node

7 is repeated; #3 is subsumed by #2; and, #1 is not completely connected. We have mapped the chromosome from GP space to clique set space. Selection of this chromosome for replacement produces the mapping back into GP space shown in Figure 4. Repair prunes dead branches of the S–expression.

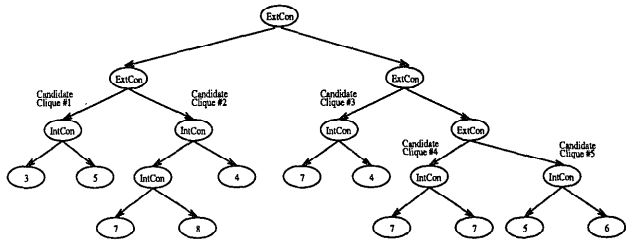

Figure 3: S–expression for 10 node graph.

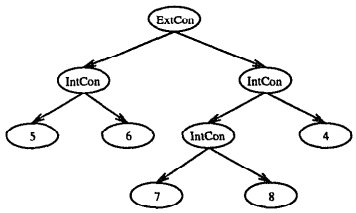

Figure 4: Repaired S–expression for 10 node graph.

Simple Repair

The extraction of candidate cliques is a repair process, and we investigate various rates of return of the repaired chromosomes into the population. Our conjecture is that the genotypes of chromosomes which succinctly capture the phenotype of the chromosome are more elegant and natural. Non–coding segments can be inserted and deleted by evolution in DNA.

The experiments use both a population size of 2000 and a generation size of 600, and are averaged over 10 trials. All statistical significance testing is done with a two-tail t-test, with a Student distribution, and a confidence level of 0.001. The 10 node graph (Figure 2) is used for clique detection. All chromosomes are repaired, and we investigate repair rates (the percentage of repaired chromosomes returned into the population) of 0%, 0.5%, 1.5%, 3%, 5%, and 10%. Repair rates greater than 0.5% (small repair rates are desirable (Orvosh & Davis 1993)) either degrade the performance or cause premature convergence, see Figure 5. Why does repair work for GA, but not for GP? Perhaps it is just this domain for which repair fails for GP. Or perhaps the repair is actually damaging the chromosome instead of fixing it up.

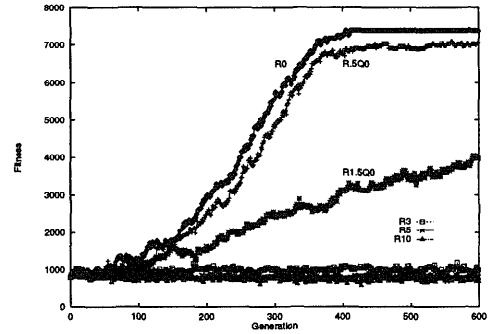

Figure 5: Best fitness for base case and repair rates of 0.5%, 1.5%, 3%, 5%, and 10%.

Repair removes "dead" or non–coding bits from the chromosome, i.e. those bits which do not contribute, either positively or negatively, to the calculation of the fitness. (In GA research, repair does not remove any bits.) Repair also removes genetic diversity. Finally, it removes any naturally occurring duplicate non–coding segments. Thus the protective backup feature of these segments is being negated. GBE research has shown that non–coding material protects building blocks from the effects of destructive crossover. We will discuss experiments in which a non–coding segments is inserted into the chromosome to investigate if there is a resulting in an increase fitness.

Repair with Duplication

Further research is performed in which the repaired chromosome is duplicated before it is thrown back into the population. For example, the chromosome represented in Figure 6 has been duplicated into the chromosome in Figure 7. While the genotypes of these two chromosome are different, the phenotypes are exactly the same, i.e. both chromosomes evaluate to the same fitness. In effect an non–coding segment has been added to the chromosome.

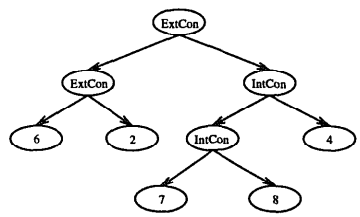

Figure 6: Best S–expression for generation 0.

Crossover is destructive for the chromosome in Figure 6: any point selected for crossover will break up a building block. Crossover cannot be completely destructive for the chromosome in Figure 7: if any point

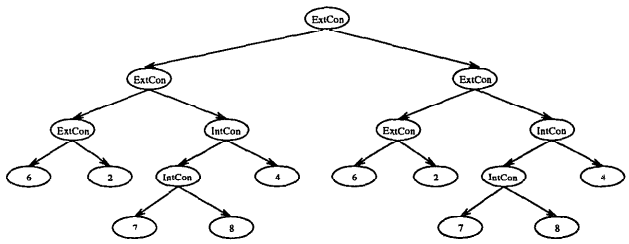

Figure 7: Generation 0's best S–expression doubled.

to the left of the root is selected for crossover, then the right subtree will remain intact. The child which "inherits" the right subtree will have a fitness greater than or equal to that of the parent. A similar argument holds for the right side. If the root is selected as the crossover point, then the child inheriting the whole tree still has a lower bound of the fitness of this parent. The non–coding segment is redundant in the parent, but it will only be redundant in the child if the other parent already contains the coding segment.

Experimental Results

The chromosome in Figure 7 should aid in the genetic search for all of the cliques in the graph, at least one of the children will be as fit as the repaired parent. The curve R0 in Figure 8 is the learning curve for the clique detector with no repairs taking place, with the solution found at about generation 354. The first experiment we conduct is to inject repairs with a 0.5% probability into the population. The curve R.5Q1 in Figure 8 is the result after adding one duplicate of the coding segment during the repair process. The solution is found at about generation 335. The hypothesis of the utility of duplication appears to not have been significant. If we examine the process, we see that if the repaired chromosome is selected for crossover, the building block should last for at least one generation. Can we force the building block to propagate through more than one generation? Yes, by adding more than one copy of the building block during repair.

We further experiment by adding three and seven copies of the coding segment. The curve R.5Q3 in Figure 8 utilizes three backups of the coding segment. The solution appears around generation 246, a significant savings of 108 generations. The curve R.5Q7 utilizes seven duplicates. The solution appears around generation 171, a savings of 183 generations. Finally, in Figure 9, we present the results for a repair rate of 10%. At a repair rate of 10% and with 7 duplicates of the coding segment, there is a significant savings of 298 generations over no repair, and 115 generations over 0.5% repair with 7 duplications.

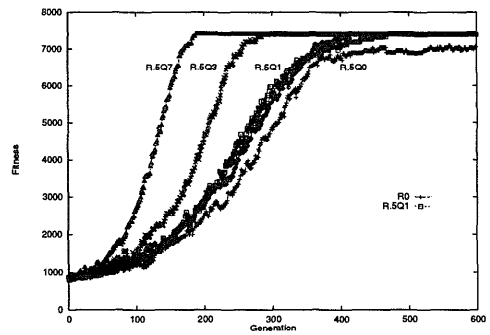

Figure 8: Best fitness for base case and a repair rate of 0.5% with 0, 1, 3, and 5 duplications.

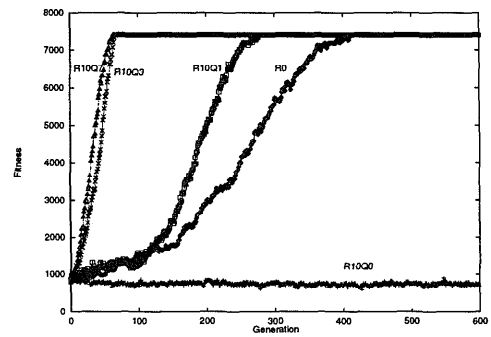

Figure 9: Best fitness for base case and a repair rate of 10% with 0, 1, 3, and 5 duplications.

We find that in general: complete removal of non–coding segments causes premature convergence; increasing duplicates of the coding segment improves the learning; and, as the repair rate increases, and more than one duplicate of the coding segment is added to the chromosome, the learning increases. This contradicts earlier findings (Orvosh & Davis 1993).

Conclusions

We utilize the tree structure of GP chromosomes to conduct experimentation into duplication of coding segments. We see that the duplication of three or more copies of the coding segments significantly speeds up the learning process for the clique detection problem. We have shown that with seven copies of the coding segment, we can at least halve the computational effort of finding the optimal solution and at best we have shown an 84% increase in finding the optimal solution over no repair and duplication at all. While the clique detection domain readily lends itself to the study of building blocks in the GP chromosome, our results are not domain dependent.

Analysis shows that this method can work for any GP domain. Simple editing rules for GP chromosomes

have been identified (Koza 1992). The methods used by compiler writers to optimize code are also applicable to "optimizing" the GP chromosome. An example of the repair and duplication process for other domains is shown in Figure 10. The parse tree to be evaluated is shown in Figure 10(a). The left subtree of the root node is *True*, which cause the middle subtree to be a coding segment and the right subtree to be a non–coding segment. The tree could be pruned, leaving only the middle subtree. The **IFTE** (IfThenElse) function can be used to add a duplicate of the coding segment (Angeline 1994), as shown in Figure 10(b).

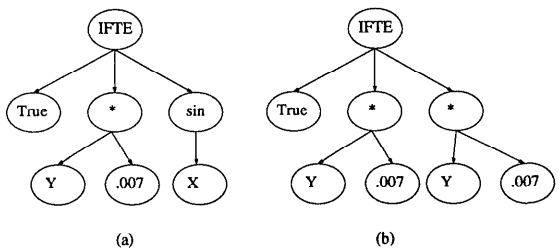

Figure 10: **IFTE** promotes duplication. (a) The right subtree of the **IFTE** node is non–coding. (b) A duplicate of the coding segment has been added.

Our function and terminal sets are not programming structures, but rather connectors for data structures. The "programming" aspect is handled by the fitness function, which is similar to how the GA fitness function translates the binary strings into the problem domains. Thus our results will also hold in domains encoded in variable-length GAs.

Repair takes place during chromosome evaluation, and thus there is no guarantee that the repaired chromosome will survive into the next generation. A new crossover function could be introduced which either takes a single parent and produces a repaired child or generates two children: one by straight selection, and the other repair. We plan to investigate this extension in a propositional inference domain. Our preliminary research shows linear non–coding segments naturally appear in this domain. Building blocks are also not as easy to identify in this domain as in the clique detection, and the optimal solution should have no duplicates of the coding segment.

References

Alberts, B.; Bray, D.; Lewis, J.; Raff, M.; Roberts, K.; and Watson, J. D. 1989. *Molecular Biology of the Cell.* Garland Publishing, Inc.

Altenberg, L. 1994. The evolution of evolvability in genetic programming. In Kinnear, Jr., K. E., ed., *Advances in Genetic Programming.* MIT Press.

Angeline, P. J. 1994. Genetic programming and emergent intelligence. In Kinnear, Jr., K. E., ed., *Advances in Genetic Programming.* MIT Press.

Futuyma, D. J. 1986. *Evolutionary Biology.* Sunderland, MA: Sinauer Associate.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning.*

Haynes, T. D.; Schoenefeld, D. A.; and Wainwright, R. L. 1996. Type inheritence in strongly typed genetic programming. In Angeline, P., and Kinnear, Jr., K. E., eds., *Advances in Genetic Programming 2.* MIT Press.

Kalmanson, K. 1986. *An Introduction to Discrete Mathematics and its Applications.* Addison–Wesley.

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

Levenick, J. R. 1991. Inserting introns improves genetic algorithm success rate: Taking a clue from biology. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 123–127. Morgan Kaufmann.

Montana, D. J. 1995. Strongly typed genetic programming. *Evolutionary Computation* 3(2):199–230.

Nordin, P. 1996. Explictly defined introns and destructive crossover in genetic programming. In Angeline, P., and Kinnear, Jr., K. E., eds., *Advances in Genetic Programming 2.* MIT Press.

O'Reilly, U.-M. 1995. *An Analysis of Genetic Programming.* Ph.D. Dissertation, Carelton University, Ottawa, Ontario, Canada.

Orvosh, D., and Davis, L. 1993. Shall we repair? Genetic algorithms, combinatorial optimization, and feasibilty constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, 650. Morgan Kaufman.

Rosca, J. 1995. Towards automatic discovery of building blocks in genetic programming. In Siegel, E. S., and Koza, J. R., eds., *AAAI Symposium on Genetic Programming*, 78–85. AAAI.

Tackett, W. A. 1993. Genetic programming for feature discovery and image discrimination. In *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93.* Morgan Kaufmann.

Wu, A. S., and Lindsay, R. K. 1995. Empirical studies of the genetic algorithm with non-coding segments. *Evolutionary Computation* 3(2).