Incorporating gene expression models into evolutionary algorithms

Michael O' Neill & Conor Ryan
Dept. of Computer Science And Information Systems
University of Limerick
Ireland
Michael.ONeill@ul.ie & Conor.Ryan@ul.ie

Abstract

We present some of the advantages gained from the incorporation of the basic principles behind gene expression into an evolutionary automatic programming system. Although we focus on advantages for the evolution of computer programs, most of these principles can also be applied to evolutionary algorithms in general.

1 Introduction

We present an example of how a primitive model of gene expression was incorporated into the Grammatical Evolution automatic programming system [Ryan et al. 98a] [Ryan et al. 98b] [Ryan & O'Neill 98] [O'Neill & Ryan 99a] [O'Neill & Ryan 99b][O'Neill & Ryan 99c] [O'Neill & Ryan 99d][O'Neill & Ryan 99e] [O'Neill et.al. 2000] [O'Neill & Ryan 2000]. As a result of this model Grammatical Evolution gains some advantages over traditional Genetic Programming [Koza 92] in which little or no gene expression exists. In recent years there has been an increase in the body of research conducted in the area of gene expression in genetic and evolutionary computation. Notable areas include genotype-phenotype distinction [Ryan et al. 98a] [Banzhaf 94] [Gruau 94], genetic-code evolution [Keller & Banzhaf 99], distributed fitness evaluation [Kargupta 99], diploidy [Ryan & Collins 98] [Goldberg 87] [Hollstein 71] [Ng & Wong 95] [Osmera et.al. 97], role of introns [Kargupta 97], degenerate genetic codes and neutral mutations [O'Neill & Ryan 99d] [Barreau 2000] [Barnett 97] [Banzhaf 94] [Vassilev & Miller 2000].

Prior to presenting Grammatical Evolution and some of the advantages it gains from the incorporation of the basic principles behind gene expression, we firstly describe the process of gene expression as elucidated by molecular biologists, before suggesting potential benefits arising from gene expression models.

2 Gene Expression Models

Molecular biology, a relatively young scientific field, has yielded many insights into the innermost workings of the fundamental units of complex organisms, cells. Discoveries include the structure of the hereditary material DNA, the discovery of proteins and their structural units, the elucidation of the genetic code, and the complex signalling networks that exist within and between cells. Of utmost importance is what has become known as *the central dogma of molecular biology*. This tenet describes the processes of transcription and translation, i.e. that DNA is transcribed to mRNA, which in turn is translated into protein. Proteins are responsible for the generation of phenotypic traits, such as eye colour, and height.

It is the genetic code, triplets of bases (a base being one of A, T, G, and C in DNA, and U replacing T in RNA), that specify each amino acid constituent of proteins. Proteins are generated during the process of translation, that literally translates the base triplets into their corresponding amino acids. There are 20 naturally occurring amino acids and 64 triplet sequences, called codons. Three of these codons are used to specify the termination of translation and do not generally specify amino acids. We are left with 61 codons for 20 amino acids. As a result it has been found that the genetic code is degenerate, that is, there is a many to one mapping where any one amino acid can be specified by many different codons. This occurs through a phenomenon known as the *wobble hypothesis* [Crick 66], and it means that a mutation at the third position in a codon does not always result in the code for a different amino acid. Such mutations are referred to as *silent* or *neutral* mutations, and these have possible implications for evolutionary search and dynamics [Kimura 83]. There has been some work recently on the evolutionary consequences of degeneracy (referred to as redundancy in this work) in natural and artificial genetic codes [Barreau 2000] [O'Neill & Ryan 99d].

An ongoing area of investigation is the regulatory processes that exist to control the expression of particular proteins from specific genes along the DNA. The discovery of special

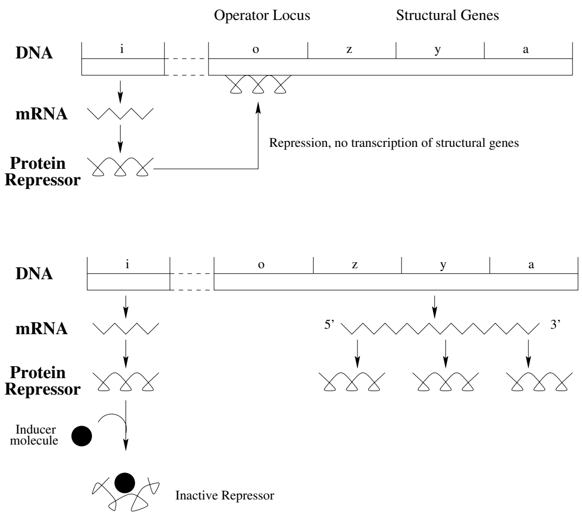

Figure 1: A diagram illustrating a modified form of the operon model as proposed by Jacob and Monod (*i* was thought to be an RNA rather than a protein). The *i* gene encodes a repressor that binds tightly to the operator *o* locus, thereby preventing transcription of the mRNA from the *z*, *y*, and *a* structural genes. When inducer is present, it combines with repressor, changing its structure so it can no longer bind to the operator locus, thus allowing translation of the three structural genes. The inducer also has the ability to remove repressor already complexed with the *o* locus.

regions along the DNA yielded the formation of the operon model [Jacob & Monod 61]. These regions known as repressors, promoters, and operators are involved in the regulation of the expression of specific genes, e.g. the infamous prokaryotic lac operon that contains three genes associated with the metabolism of lactose. An overview of the operon model can be seen in Figure 1.

What is crucial to note when dealing with gene expression, is that it is occurring within a cellular environment with complex feedback loops controlling the further expression of genes. While all cells in an organism have identical genetic material each cell has the ability to specialise in terms of functionality and structure depending on what genes are expressed at certain stages of cell development. In [Kennedy 98] a novel model of a biological cell is presented in which each cell has a genome and metabolism. Pairs of these genomes and metabolisms co-evolve using a genetic algorithm. A simple genomic language is introduced that follows Jacob and Monod's operon model [Jacob & Monod 61], such that genes can be placed at any loci on the genome. Thus a model is generated in which the genome specifies proteins such as enzymes which can catalyse metabolic reactions, and the chemicals of metabolism have the ability to regulate genes and allow expression of proteins from the genome. The model, partly Lamarckian due to the manner in which the metabolism evolves, is found to be more efficient at solving the bootstrapping problem due to the co-evolution of the metabolism with the genome.

In the next section we will present some of the potential advantages to be gained from incorporating gene expression into our algorithms.

3 Potential Advantages arising from Gene Expression

Potential advantages arising from gene expression include:

1. A separation of search and solution spaces (genotype-phenotype distinction)
2. Genetic diversity could be maintained during runs through the distinction of genotype and phenotype particularly in the case of point 3
3. Preservation of functionality while allowing continuation of search.
4. Generalised encoding that can represent a variety of structures without the need for specially engineered genetic operators - functional diversity of proteins.
5. Degenerate Encodings - efficiency gains in terms of evolutionary search, e.g. through the use of neutral mutations and their corresponding neutral networks, for example see [Barnett 97] [Newman & Engelhardt 1998].
6. Compression of representation - large structures represented with a relatively small genotype.
7. Alternative implementation of functions, for example, through the use of the operon model.
8. Positional Independence - this amounts to the position (locus) of a gene on the chromosome bearing no effect on its functionality. This useful phenomenon could be achieved using the operon model by allowing metabolites to recognise a specific promoter region on the chromosome thus switching on or off that particular gene (or set of genes) expression. This could also be achieved through the use of novel expression mechanisms, some alternatives are currently being investigated by the authors.

4 Grammatical Evolution - An example of primitive gene expression modeling

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve code in any language. Individuals are comprised of variable-length binary strings, and a genotype-to-phenotype mapping process is employed to generate the output program. We use this mapping process to ensure the generation of syntactically correct programs by utilising a language definition, namely a Backus Naur Form (BNF) grammar. In conjunction with a wrapping operator, the grammar ensures the generation of legal phenotypes for the main. For a discussion of various approaches to the generation of legal phenotypes from genotypes we refer the reader to [Yu & Bentley 98].

When tackling any problem with GE a suitable BNF definition must first be decided upon. The BNF can be either the specification of an entire language, or perhaps more usefully a subset of a language geared towards the problem at hand. For example, a BNF for the Santa Fe Ant Trail [Langdon & Poli 98] [O'Neill & Ryan 99a] could be;

```
N = {<code>,<line>,<condition>,<op>}

T = {left(),right(),move(),food_ahead(),
       else,if,{,},(,)}

S = <code>
```

And P can be represented as:

```
(1) <code> :: = <line>                  (A)
              | <code><line>            (B)

(2) <line> :: = <condition>             (A)
              | <op>                    (B)

(3) <condition> :: = if(food_ahead()){
                        <line>
                     }
                     else{
                        <line>
                     }

(4) <op> :: =     left();                 (A)
               |  right();                (B)
               |  move();                 (C)
```

where the operations `left()`, `right()`, `move()`, and `food_ahead()`, are all functions written in the C programming language. N is the set of non-terminals, T, the set of terminals, P, a set of production rules that map the elements of N to T, and S is a start symbol which is a member of N.

The genotype is then used to map the start symbol onto terminals by reading codons of 8 bits to generate a corresponding integer value, from which an appropriate production rule is selected. A rule is selected by using the following,

$$Rule = (Codon\ Integer\ Value)$$

$$MOD$$

$$(Number\ of\ Rule\ Choices\ for\ the\ current\ N)$$

Considering rule #4 from above, i.e. given the non-terminal <op> there are three production rules to select from. If we assume the codon being read produces the integer 6, then $6\ MOD\ 3 = 0$ would select rule `(A)left()`. Each time a production rule has to be selected to map from a non-terminal, another codon is read, and in this way, the system traverses the genome.

During the genotype to phenotype mapping process it is possible for individuals to run out of codons, and in this case we wrap the individual, and reuse the codons. This is quite an unusual approach in EA's, as it is entirely possible for certain codons to be used two or more times. In GE, each time the same codon is expressed it will always generate the same integer value, but depending on the current non-terminal, may have a different effect, that is, it may select a different production rule. What is crucial, however, is that each time a particular individual is mapped from its genotype to its phenotype, the same output is generated. This is because the same choices are made each time. It is possible that an incomplete mapping could occur, even after wrapping, and in this event the individual in question is given the lowest fitness value possible. The selection and replacement mechanisms should operate accordingly to increase the likelihood that these invalid individuals are removed from the population.

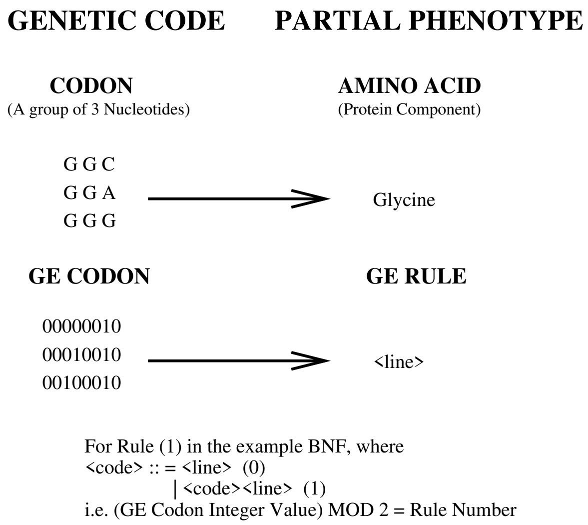

Figure 2: A comparison between genetic code degeneracy in molecular biology and Grammatical Evolution, note how a single point mutation at the third base in the biological genetic code will always result in the selection of the amino acid Glycine. Amino acids are the components of proteins which are responsible for phenotypic traits. In Grammatical Evolution we give an example of how a single bit mutation in a codon can result in the application of the same production rule $< line >$. The production rules in Grammatical Evolution are combined in such a way as to produce a syntactically correct program in the language specified by the grammar.

An incomplete mapping could arise if the integer values expressed by the genotype were applying the same production rules over and over. For example, given an individual with three codons, all of which specify rule 0 from below,

```
(1) <expr> :: = <expr><op><expr>      (0)
              | (<expr><op><expr>)    (1)
              | <preop>(<expr>)       (2)
              | <var>                 (3)
```

even after wrapping the mapping process would be incomplete and would carry on indefinitely unless stopped. This occurs because the nonterminal `<expr>` is being mapped indefinitely by production rule 0, i.e., it becomes `<expr><op><expr>`. Therefore, the leftmost `<expr>` after each application of a production would itself be mapped to a `<expr><op><expr>`, resulting in an expression continually growing as follows: `<expr><op><expr><op><expr><op><expr>` etc. Such an individual is dubbed "invalid" as it will never undergo a complete mapping to a set of terminals.

To reduce the number of invalid individuals being passed from generation to generation a steady state replacement mechanism is employed. A consequence of the steady state method is its tendency to maintain fit individuals at the expense of less fit, and in particular, invalid individuals.

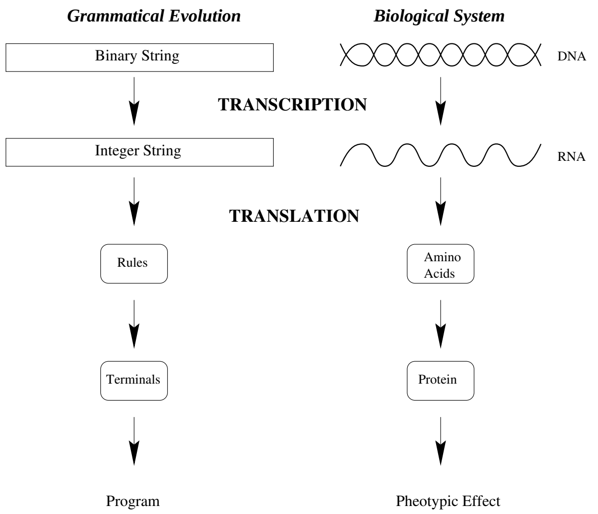

Figure 3: A comparison between Grammatical Evolution and molecular biology which shows the mapping process from genotype to phenotype.

Results presented in [O'Neill & Ryan 99d] show that the degenerate genetic code employed in GE also plays a role in preserving individual validity. With such a representation scheme, different codes can represent the same symbol, or in GE, the same production rule. This allows modifications to occur to the genotype without affecting the phenotype, or functionality. As mentioned earlier, mutation events that do not alter the phenotypic fitness are termed *neutral mutations* [Kimura 83] [Banzhaf 94]. A degenerate coding mechanism is also employed in nature and a comparison between GE's genetic code and that of molecular biology can be seen in Figure 2.

An overview of how GE operates in comparison with a biological gene expression system can be seen in Figure 3. It can be seen that genes are expressed as proteins that produce a phenotype. More than one protein may be responsible for producing any one phenotype, and a similar situation occurs in GE. Here a codon results in the selection of rules that map non-terminals onto either more non-terminals, or terminals, it is a combination of these terminals and non-terminals that produce the phenotype, or program.

5 An analysis of Grammatical Evolution

Given the description of GE in the previous section it is apparent that we have utilised a very crude model of biological gene expression, i.e. the genotype-phenotype distinction, the degenerate genetic code, and the wrapping operator. There are no gene expression controls such as those in existence in operons. With respect to the potential advantages arising from gene expression we can say that GE uses the first six.

We have a genotype-phenotype distinction such that our genotype representation is independent of the output code. We can therefore take advantage of advances in the field of evolutionary algorithms and indeed other search techniques. In particular, we use a simple variable-length GA. It has been shown that standard GA's do not scale very well for certain problems. However, there has been a proliferation of research in this area that has given rise to enhanced versions of the simple GA such as the Gene Expression Messy GA [Kargupta 96], the Linkage Learning GA [Harik 97], and the Extended Compact Genetic Algorithm [Harik 99]. GE is in a position to be able to adopt these advances in genetic algorithm research with only minor changes.

Advantages two and five, due to degenerate encodings and genetic diversity, have been shown to exist in GE [O'Neill & Ryan 99c] [O'Neill & Ryan 99d]. As reported the degenerate genetic code has various advantages, such as a preservation of individual functionality in spite of genetic variation, an improvement in the algorithms success rate in terms of the time taken to find solutions for the problems analysed, and a maintenance of genetic diversity within the population over the course of runs.

Advantage three, preservation of functionality, is largely achieved through the degenerate encoding as reported in [O'Neill & Ryan 99d]. It is also aided by the wrapping operator [O'Neill & Ryan 99c] in conjunction with the steady state replacement strategy.

Compression of representation (advantage six) can be achieved in GE, rather than representing the entire output program in our genotype, we search the transformations to generate the program in terms of the production rules that are applied during the mapping process. The wrapping operator has also been shown to play a role in the compression of genotype string length [O'Neill & Ryan 99c]. Results have suggested that GE does not suffer from bloat to the same extent as GP.

As for alternative function implementation (advantage seven) an operon-like model has not been implemented in standard GE to-date. We have, however, utilised the grammar and thus the mapping process to achieve function implementation in terms of grammar-based automatic function definition [O'Neill & Ryan 2000] with encouraging results.

Positional independence (advantage eight) has yet to be exploited in GE. It is envisaged that the use of operon-like structures could be used to allow function to be independent of locus. This could be achieved by allowing the use of operator/promotor regions and start/stop codons that would allow a function to be turned on given a certain signal, no matter what its current locus is.

6 Conclusions

We have presented gene expression models brought to light by researchers in the field molecular biology, and described some of the potential advantages to utilising such a model in genetic and evolutionary computation. We show how a crude model of gene expression was incorporated in Grammatical Evolution, an automatic programming system, and reported some of the advantages arising from this model.

We can conclude that the incorporation of gene expression models into the field of genetic and evolutionary computation presents us with definite advantages as illustrated by results using the Grammatical Evolution system. Perhaps further modeling of gene expression models, such as the use of operons, could yield more impressive results and insights into evolutionary processes and the computational nature of gene expression.

References

[Banzhaf 94] Banzhaf, W. 1994. Genotype-Phenotype Mapping and Neutral Variation - A case study in Genetic Programming. In *Parallel Problem Solving from Nature III*. Springer.

[Barnett 97] Barnett, Lionel. Evolutionary dynamics on fitness landscapes with neutrality. Master's thesis, School of Cognitive Sciences, University of East Sussex, Brighton, 1997.

[Barreau 2000] Barreau G. The Evolutionary Consequences of Redundancy in Natural and Artificial Genetic Codes. Ph.D. Thesis, University of Sussex, 2000.

[Crick 66] Crick F.H.C. Codon-anticodon pairing; the wobble hypothesis. *Journal of Molecular Biology*. 19:548-555, 1966.

[Goldberg 87] Goldberg, D. 1987. Nonstationary function optimisation with dominance and diploidy. In *Proceedings of the Second International Conference on Genetic Algorithms*.

[Gruau 94] Gruau F. Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm. Ph.D. Thesis, 1994.

[Harik 99] Harik G.R. Linkage Learning via Probabilistic Modeling in the ECGA. IlliGAL Report 99010.

[Harik 97] Harik G.R. Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms. PhD dissertation, University of Michigan, Ann Arbor, Michigan. Also IlliGAL Report 97005.

[Hollstein 71] Hollstein R. B. 1971. Artificial genetic adaptation in computer control systems. PhD dissertation, University of Michigan.

[Jacob & Monod 61] Jacob F., and Monod J. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology*. 3:318-356, 1961.

[Kargupta 99] Kargupta H., and Sarkar K. 1999. Function Induction, Gene Expression, and Evolutionary Representation Construction. In *Proceedings of GECCO'99: The Genetic and Evolutionary Computation Conference 1999*, vol. 1, pages 313-320.

[Kargupta 97] Kargupta, H. 1997. Relation Learning In Gene Expression: Introns, Variable Length Representation, And All That. Presented in the Workshop on Exploring Non-Coding Segments and Genetic-Based Encodings. International Conference On Genetic Algorithms, Michigan.

[Kargupta 96] Kargupta, H. 1996. The gene expression messy genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computation 1996*, pages 814-819, Piscataway, New Jersey.

[Keller & Banzhaf 99] Keller R., Banzhaf W. 1999. The evolution of genetic code in genetic programming. In *Proceedings of GECCO'99: The Genetic and Evolutionary Computation Conference 1999*, vol.2, pages 1077-1082.

[Kennedy 98] Kennedy Paul J. 1998. Simulation of the Evolution of Single Celled Organisms with Genome, Metabolism and Time-Varying Phenotype. Ph.D. Thesis, University of Technology, Sydney.

[Kimura 83] Kimura, M. 1983. The Neutral Theory of Molecular Evolution. Cambridge University Press.

[Koza 92] Koza, J. 1992. *Genetic Programming*. MIT Press.

[Langdon & Poli 98] Langdon, W. & Poli, R. Why Ant's Are Hard. In *Proceedings of Genetic Programming 1998*, pages 193-201 .

[Newman & Engelhardt 1998] Newman, M.E.J., Engelhardt, Robin. Effects of neutral selection on the evolution of molecular species. *Proc. R. Soc. London B*, 1998.

[Ng & Wong 95] Ng, K. and Wong, K. 1995. A new diploid scheme and dominance change mechanism for non-stationary function optimisation. In *Proceedings of the Fifth International Conference on Genetic Algorithms*.

[Osmera et.al. 97] Osmera, P., Kvasnicka, V., and Pospichal, J. 1997. Genetic Algorithms with Diplod Chromosomes. In *Proceedings of Mendel'97: 3rd International Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks and Rough Sets*, Brno, Czech Republic.

[O'Neill et.al. 2000] O'Neill M., Collins J.J., and Ryan C. Automatic Generation of Robot Behaviours using Grammatical Evolution. In *Proceedings of AROB 2000: The Fifth International Symposium on Artificial Life and Robotics*, Oita, Japan.

[O'Neill & Ryan 2000] O'Neill M., Ryan C. Crossover in Grammatical Evolution: A Smooth Operator? In *Proceedings of the Third European Workshop on Genetic Programming 2000*.

[O'Neill & Ryan 2000] O'Neill M., Ryan C. Grammar based function definition in Grammatical Evolution. In *Proceedings of GECCO 2000: The Genetic and Evolutionary Computation Conference 2000*. To appear.

[O'Neill & Ryan 99a] O'Neill M., Ryan C. Evolving Multi-line Compilable C Programs. In *Proceedings of the Second European Workshop on Genetic Programming 1999*.

[O'Neill & Ryan 99b] O'Neill M., Ryan C. Automatic Generation of Caching Algorithms. In *Proceedings of EUROGEN'99: A short course on Evolutionary Computation*, Jyvaskyla, Finland, pages 127-134.

[O'Neill & Ryan 99c] O'Neill M., Ryan C. Under the Hood of Grammatical Evolution. In *Proceedings of GECCO'99: The Genetic and Evolutionary Computation Conference 1999*, Vol. 2, pp. 1143-1148.

[O'Neill & Ryan 99d] O'Neill M., Ryan C. Genetic Code Degeneracy: Implications for Grammatical Evolution and Beyond, in *ECAL'99: Proceedings of the Fifth European Conference on Artificial Life*, Lausanne, Switzerland.

[O'Neill & Ryan 99e] O'Neill M., Ryan C. Steps Towards Khepera Dance Improvisation. In *Proceedings of the First International Khepera Workshop 1999*.

[Ryan & Collins 98] Ryan, C. and Collins, J.J. 1998. Polygenic Inheritance - A Haploid Scheme that Can Outperform Diploidy. In *Procs. of the Fifth Int. Conf. on Parallel Problem Solving from Nature (PPSN V)*. September 1998, Amsterdam, pages 178-187. Springer LNCS 1498.

[Ryan et al. 98a] Ryan C., Collins J.J., O'Neill M. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*, pages 83-95 . Springer-Verlag.

[Ryan et al. 98b] Ryan C., O'Neill M., Collins J.J. 1998. Grammatical Evolution: Solving Trigonometric Identities. In *Proceedings of Mendel '98: 4th International Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks and Rough Sets*, Brno, Czech Republic. Pages 111-119.

[Ryan & O'Neill 98] Ryan C., O'Neill M. Grammatical Evolution: A Steady State Approach. In *Late Breaking Papers, Genetic Programming 1998*, pages 180-185.

[Vassilev & Miller 2000] Vassilev V.K., Miller J. F. The Advantages of Landscape Neutrality in Digital Circuit Evolution. In *Proceedings of ICES 2000. Evolvable Systems: From Biology to Hardware*, Edinburgh, Scotland. Pages 252-263.

[Yu & Bentley 98] Yu, T. and Bentley, P. 1998 Methods to Evolve Legal Phenotypes. In *Lecture Notes in Computer Science 1498, Proceedings of Parallel Problem Solving from Nature V*. Pages 280-291. Springer.