Discovery of the Boolean Functions to the Best Density-Classification Rules Using Gene Expression Programming

Cândida Ferreira

Gepsoft, 37 The Ridings,
Bristol BS13 8NU, UK
candidaf@gepsoft.com
http://www.gepsoft.com

In E. Lutton, J. A. Foster, J. Miller, C. Ryan, and A. G. B. Tettamanzi, eds., Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002, volume 2278 of Lecture Notes in Computer Science, pages 51-60, Springer-Verlag, Berlin, Germany, 2002.

Cellular automata are idealized versions of massively parallel, decentralized computing systems capable of emergent behaviors. These complex behaviors result from the simultaneous execution of simple rules at multiple local sites. A widely studied behavior consists of correctly determining the density of an initial configuration, and both human and computer-written rules have been found that perform with high efficiency at this task. However, the two best rules for the density-classification task, Coevolution_1 and Coevolution_2, were discovered using a coevolutionary algorithm in which a genetic algorithm evolved the rules and, therefore, only the output bits of the rules are known. However, to understand why these and other rules perform so well and how the information is transmitted throughout the cellular automata, the Boolean expressions that orchestrate this behavior must be known. The results presented in this work are a contribution in that direction.

1. Introduction

Genetic programming (GP) evolves computer programs by genetically modifying nonlinear entities with different sizes and shapes [1]. These nonlinear entities can be represented as diagrams or trees. Gene expression programming (GEP) is an extension to GP that also evolves computer programs of different sizes and shapes, but these programs are encoded in a linear chromosome of fixed length [2]. One strength of the GEP approach is that the creation of genetic diversity is extremely simplified as genetic operators work at the chromosome level. Indeed, due to the structural organization of GEP chromosomes any modification made in the genome results always in valid programs. Another strength of GEP consists of its unique, multigenic nature which allows the evolution of more complex programs composed of several sub-programs.

Cellular automata (CA) have been studied widely as they are idealized versions of massively parallel, decentralized computing systems capable of emergent behaviors (for overviews of CA theory and applications see, e.g., [3, 4]). These complex behaviors result from the simultaneous execution of simple rules at multiple local sites. In the density-classification task, a simple rule involving a small neighborhood and operating simultaneously in all the cells of a one-dimensional cellular automaton, should be capable of making the CA con-

verge into a state of all 1's if the initial configuration (IC) has a higher density of 1's, or into a state of all 0's if the IC has a higher density of 0's.

The challenging problem of density-classification started with the Gacs-Kurdyumov-Levin rule (GKL rule) [5], designed in 1978 by hand to study reliable computation and phase transitions in one-dimensional spatially extended systems. Although not especially designed for the density-classification task, the GKL rule was for some time the rule with best performance on this task [6]. In 1993, Lawrence Davis obtained a new rule modifying the GKL rule [7]. This new rule, Davis rule, achieved an accuracy slightly better than the GKL rule. Similarly, Rajarshi Das cleverly modified rules discovered by genetic algorithms (GAs), obtaining a new rule (Das rule) that performed slightly better than the GKL rule and the Davis rule [7]. Genetic programming discovered a new rule (GP rule) slightly better than the previous rules [7]. Gene expression programming discovered two new rules (GEP_1 and GEP_2 rules) better than all the previous rules [2]. And finally, Juillé and Pollack [8], using coevolutionary learning, discovered two new rules ($Coevolution_1$ and $Coevolution_2$) significantly better than all the previous rules.

However, the two best rules, $Coevolution_1$ and $Coevolution_2$, were discovered using a coevolutionary approach in which a GA evolved the rules and, therefore, only the output bits of the rules are known. However, if we want to understand why these rules perform so well and how the information is transmitted throughout the CA, it is mandatory to know the Boolean expressions that orchestrate the behavior of the CA. In this work, GEP is successfully applied to find the Boolean expressions behind the two best density-classification rules, providing a useful resource for future research on emergent behavior.

2. Genetic Algorithms

Gene expression programming belongs to a wider group of genetic algorithms as it uses populations of individuals, selects individuals according to fitness, and introduces genetic variation using one or more genetic operators [2, 6].

2.1. Fundamental Classes of Genetic Algorithms

Structurally, genetic algorithms can be subdivided in three fundamental groups: i) Genetic algorithms with individuals consisting of linear chromosomes of fixed length devoid of complex expression. In these systems, replicators (chromosomes) survive by virtue of their own properties. The algorithm invented by Holland [9] belongs to this group, and is known as genetic algorithm or GA; ii) Genetic algorithms with individuals consisting of ramified structures of different sizes and shapes and, therefore, capable of assuming a richer number of functionalities. In these systems, replicators (ramified structures) also survive by virtue of their own properties. The algorithm invented by Cramer [10] and later developed by Koza [1] belongs to this group and is known as genetic programming or GP; iii) Genetic algorithms with individuals encoded as linear chromosomes of fixed length which are afterwards expressed as ramified structures of different sizes and shapes. In these systems, replicators (chromosomes) survive by virtue of causal effects on the phenotype (ramified structures). The algorithm invented by myself [2] belongs to this group and is known as gene expression programming or GEP.

The fact that GEP shares with GP the same kind of ramified structure, shows that GEP can be used, for one thing, to retrace easily the steps undertaken by GP and, for another, to explore easily new frontiers opened up by the crossing of the phenotype threshold. Next follows a brief introduction to gene expression programming.

2.2. Gene Expression Programming

The phenotype of GEP individuals consists of the same kind of diagram representations used by GP. However, these complex entities are encoded in simpler, linear structures of fixed length - the chromosomes. Thus, the main players in GEP are two entities: the chromosomes and the ramified structures or expression trees (ETs), being the latter the expression of the genetic information encoded in the former. The process of information decoding (from the chromosomes to the ETs) is called translation. And this translation implies obviously a kind of code and a set of rules. The genetic code is very simple: a one-to-one relationship between the symbols of the chromosome and the functions or terminals they represent. The rules are also very simple: they determine the spatial organization of the functions and terminals in the ETs and the type of interaction between sub-ETs in multigenic systems.

In GEP there are therefore two languages: the language of the genes and the language of ETs and, in this simple replicator/phenotype system, knowing the sequence or structure of one, is knowing the other. In nature, although the inference of the sequence of proteins given the sequence of genes and *vice versa* is possible, practically nothing is known about the rules that determine the three-dimensional structure of proteins. But in GEP, thanks to the simple rules that determine the structure of ETs and their interactions, it is possible to infer exactly the phenotype given the sequence of a gene, and *vice versa*. This bilingual and unequivocal system is called *Karva* language. The details of this language are given below.

2.2.1. Open Reading Frames and Genes

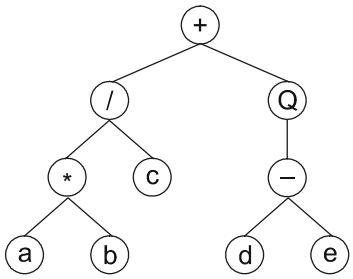
In GEP, the genome or chromosome consists of a linear, symbolic string of fixed length composed of one or more genes. Despite their fixed length, GEP chromosomes code for ETs with different sizes and shapes, as will next be shown.

The structural organization of GEP genes is better understood in terms of open reading frames (ORFs). In biology, an ORF or coding sequence of a gene begins with the “start” codon, continues with the amino acid codons, and ends at a termination codon. However, a gene is more than the respective ORF, with sequences upstream of the start codon and sequences downstream of the stop codon. Although in GEP the start site is always the first position of a gene, the termination point not always coincides with the last position of a gene. It is common for GEP genes to have non-coding regions downstream of the termination point. (For now these non-coding regions will not be considered because they do not interfere with the product of expression.)

Consider, for example, the algebraic expression:

$$\frac{a \cdot b}{c} + \sqrt{d - e} \tag{1}$$

It can also be represented as a diagram:



where “Q” represents the square root function.

This kind of diagram representation is in fact the phenotype of GEP chromosomes, being the genotype easily inferred from the phenotype as follows:

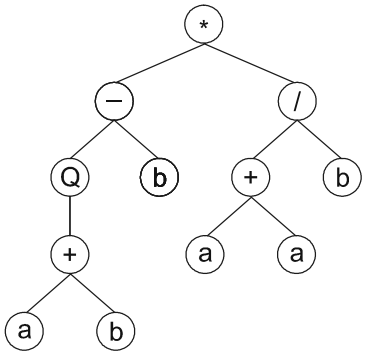
```
0123456789
+/Q*c-abde                                                    (2)
```

which is the straightforward reading of the ET from left to right and from top to bottom (exactly as we read a page of text). The expression (2) is an ORF, starting at "+" (position 0) and terminating at "e" (position 9). These ORFs are called K-expressions (from Karva notation).

Consider another ORF, the following K-expression:

```
012345678901
*-/Qb+b+aaab                                                 (3)
```

Its expression as an ET is also very simple and straightforward. To express correctly the ORF, the rules governing the spatial distribution of functions and terminals must be followed. The start position (position 0) in the ORF corresponds to the root of the ET. Then, below each function are attached as many branches as there are arguments to that function. The assemblage is complete when a baseline composed only of terminals (the variables or constants used in a problem) is formed. So, for the K-expression (3) above, the following ET is formed:



Looking at the structure of GEP ORFs only, it is difficult or even impossible to see the advantages of such a representation, except perhaps for its simplicity and elegance. However, when ORFs are analyzed in the context of a gene, the advantages of this representation become obvious. As stated previously, GEP chromosomes have fixed length, and they are composed of one or more genes of equal length. Therefore the length of a gene is also fixed. Thus, in GEP, what varies is not the length of genes but the length of the ORFs. Indeed, the length of an ORF may be equal to or less than the length of the gene. In the first case, the termination point coincides with the end of the gene and, in the last case, the termination point is somewhere upstream of the end of the gene.

As will next be shown, the non-coding sequences of GEP genes are extremely important for they allow the modification of the genome using any genetic operator without restrictions, producing always syntactically correct programs. The section proceeds with the study of the structural organization of GEP genes in order to show how these genes invariably code for syntactically correct programs and why they allow the unconstrained application of numerous genetic operators.

2.2.2. Structural Organization of Genes

GEP genes are composed of a head and a tail. The head contains symbols that represent both functions and terminals, whereas the tail contains only terminals. For each problem, the

length of the head *h* is chosen, whereas the length of the tail *t* is a function of *h* and maximum arity *n*, and is evaluated by the equation:

$$t = h \cdot (n - 1) + 1 \tag{4}$$

Consider a gene for which the set of functions consists of F = {Q, *, /, -, +} and the set of terminals T = {a, b}. In this case, $n = 2$; and if an $h = 15$ is chosen, then $t = 16$. Thus, the length of the gene *g* is 15+16=31. One such gene is shown below (the tail is shown in bold):

```
01234567890123456789012345678 90
/aQ/b*ab/Qa*b*-ababaababbabbbba                (5)
```

It codes for an ET with eight nodes. Note that, in this case, the ORF ends at position 7 whereas the gene ends at position 30.

Suppose now a mutation occurred at position 2, changing the "Q" into "+". Then the following gene is obtained:

```
01234567890123456789012345678 90
/a+/b*ab/Qa*b*-ababaababbabbbba                (6)
```

In this case, its expression gives a new ET with 18 nodes. Note that the termination point shifts 10 positions to the right (position 17).

Obviously the opposite might also happen, and the ORF can be shortened. For example, consider again gene (5) above, and suppose a mutation occurred at position 5, changing the "*" into "b", obtaining:

```
01234567890123456789012345678 90
/aQ/bbab/Qa*b*-ababaababbabbbba                (7)
```

Its expression results in a new ET with six nodes. Note that the ORF ends at position 5, shortening the parental ET in two nodes.

So, despite their fixed length, GEP genes have the potential to code for ETs of different sizes and shapes, being the simplest composed of only one node (when the first element of a gene is a terminal) and the biggest composed of as many nodes as the length of the gene (when all head elements are functions with maximum arity).

It is evident from the examples above, that any modification made in the genome, no matter how profound, results always in a structurally correct ET as long as the structural organization of genes is maintained. Indeed, the implementation of high-performing genetic operators in GEP is a child's play, and Ferreira [2] describes seven: point mutation, RIS and IS transposition, two- and one-point recombination, gene transposition and gene recombination.

2.2.3. Multigenic Chromosomes

GEP chromosomes are usually composed of more than one gene of equal length. For each problem or run, the number of genes, as well as the length of the head, are *a priori* chosen. Each gene codes for a sub-ET and the sub-ETs interact with one another forming a more complex multi-subunit ET.

Consider the following chromosome with length 45, composed of three genes:

```
012345678901234012345678901234012345678901234
Q/*b+Qababaaba-abQ/*+bababbab**-*bb/babaaaab   (8)
```

It has three ORFs, and each ORF codes for a sub-ET. Position 0 marks the start of each gene.

The end of each ORF, though, is only evident upon construction of the respective sub-ET. In this case, the first ORF ends at position 8; the second ORF ends at position 2; and the last ORF ends at position 10. Thus, GEP chromosomes are composed of one or more ORFs, each ORF coding for a structurally and functionally unique sub-ET. Depending on the problem at hand, the sub-ETs encoded by each gene may be selected individually according to their respective fitness (for example, in problems with multiple outputs), or they may form a more complex, multi-subunit ET where individual sub-ETs interact with one another by a particular kind of posttranslational interaction or linking. For instance, algebraic sub-ETs are usually linked by addition or multiplication whereas Boolean sub-ETs are usually linked by OR, AND, or IF.

3. The Density-Classification Task

The simplest CA is a wrap-around array of N binary-state cells, where each cell is connected to r neighbors from both sides. The state of each cell is updated by a defined rule. The rule is applied simultaneously in all the cells, and the process is iterated for t time steps. In the most frequently studied version of the density-classification task, $N = 149$ and $r = 3$. The central cell is represented by "u"; the three cells to the left are represented by "c", "b", and "a"; and the three cells to the right are represented by "1", "2", and "3".

The task of density-classification consists of correctly determining whether ICs contain a majority of 1's or a majority of 0's, by making the system converge, respectively, to an all 1's state, or to a state of all 0's. As the density of an IC is a function of N arguments, the actions of local cells with limited information and communication must be coordinated with one another in order to classify correctly the ICs. Indeed, to find by hand, in a search space of 2^{128} transition rules, CA rules that perform well is an almost impossible task and, therefore, several evolutionary algorithms were used to evolve better rules than the GKL rule [2, 7, 8, 11, 12, 13]. The best rules with performances of 86.0% (Coevolution_2) and 85.1% (Coevolution_1) were discovered using a coevolutionary approach between ICs and rules evolved by a GA [8]. The Boolean expressions of these rules are not known and little or no knowledge can be extracted from the output bits of their rule tables. In the next section, GEP is used to find the solutions to these rules.

4. Discovering the Boolean Functions to the Best Rules for the Density-Classification Task

The space of possible rules for Boolean functions of seven arguments is the huge space of $2^{2^7} = 2^{128}$ rules, and the size of the space of possible computer programs that can be composed using the elements of the function and terminal sets is greater still. Therefore, the discovery of the Boolean functions to the Coevolution_1, and Coevolution_2 rules is no trivial task and, in fact, their discovery by GEP involved several optimization runs where the best solution of a run was used as seed to evolve better programs. This kind of strategy is inevitable while trying to solve real-world problems of great complexity. Good intermediate solutions are some times hard to find, and start everything anew is no guarantee that a perfect solution or even a better intermediate solution will be found. Therefore, it is advantageous to use a good intermediate solution as seed to start a new evolutionary cycle, with the advantage that the seed or the evolutionary conditions can be slightly changed before an optimization. For instance, we can increase the gene length, introduce or remove a neutral gene, introduce or remove new functions in the function set, linearize a multigenic solution, change the fitness function or the selection environment, change the population size, change the

genetic operators or their rates, and so forth. The particular evolutionary strategy followed in each case is given below.

4.1. Experimental Setup

The total $2^7 = 128$ transition states of each rule consists of the fitness cases used to evaluate fitness. In all cases, the set of functions F = { N, A, O, X, D, R, I, M} (representing, respectively, NOT, AND, OR, XOR, NAND, NOR, IF, and Majority, where the first is a function of one argument, the second through fifth are functions of two arguments, and the last two are functions of three arguments), and the set of terminals T = {c, b, a, u, 1, 2, 3}. The fitness f_i of a program i is evaluated by the equation:

$$\text{If } n \geq \tfrac{1}{2} C_t, \text{ then } f_i = n; \text{ else } f_i = 1 \tag{9}$$

where n is the number of fitness cases correctly evaluated, and C_t is the total number of fitness cases. Thus, for the entire set of fitness cases, $f_{\max} = 128$.

4.2. Coevolution$_1$ Rule

The Coevolution$_1$ rule has a performance of 85.1% and was discovered using a coevolutionary approach between GA-evolved rules and ICs [8].

Gene expression programming found the solution to the Coevolution$_1$ rule in four phases. Firstly, chromosomes composed of nine genes of length 22 each were used. The sub-ETs were linked by IF, forming a huge multi-subunit ET. An intermediate solution with fitness 121 was found after 47093 generations of a small initial population of 30 individuals. Then, this intermediate solution was used as seed, and after 11785 generations a better solution with fitness 123 was found. Then, the genes of this intermediate solution were increased from 22 to 34 and populations of 50 individuals were left to evolve for 50000 generations. On generation 37362 an intermediate solution with fitness 127 was discovered. Finally, this program was used as seed to create another population, this time with 50 individuals. After 15189 generations the following perfect solution to the Coevolution$_1$ rule was found:

```
MAD21RMDuOa1ab1ucb31bbu1acc31bu1c1
XXM1ROM3Rubaaa3322c2ba13abcu3c1uc1
NOaANMbXIXbb3buc33bc2bba3b21u11uc1
XRRID1ADbDA2cb1bcuu1cbu133u1aaaa3c
AOaOIAcONuDbcca33auauba3332b11b2bb
MXIODuRXOA31122b3cb3a3bau1cbbbb312
MuIADIOXINMa1cua1uu1bu2cab1cuccc2a
NNIMOODR1ODuu31u132u12babc1bu2aub2
IuIIRDOXIMb2cu312ua23a2c3222ba2ab3
```

where the sub-ETs are linked 3-by-3 with IF, forming three big clusters that are, in their turn, linked also 3-by-3 with IF.

4.3. Coevolution$_2$ Rule

The Coevolution$_2$ rule has a performance of 86.0% and was also discovered using a coevolutionary approach between GA-evolved rules and ICs [8].

The solution to the Coevolution$_2$ rule was easier to find than the solution to the previous rule. For this problem, populations of 50 individuals with chromosomes composed of 9 genes of length 22 each were used. The sub-ETs were linked also by the IF function.

In one run, an intermediate solution with fitness 126 was found by generation 18660. This program was used as seed to initialize another evolutionary epoch. After 23807 generations the following solution was found:

```
X3ONOD2acaa1332baa3211
R1XuION1ua33aa3321cbc2
RMaDAM3u1bb13cuuc2bubu
RRRA22X23b31a3a3122aab
M3D33NM2b21u22aa31bc2b
AOAII3ac2a3c2ua2u21u33
DXO1DXI1ab23u11ba1bba1
IMXcDMR1ub1bcua231cuu1
MDIMAMO2ubac212c22ccac
```

which is a perfect solution to the Coevolution$_2$ rule. The sub-ETs encoded by each gene are linked 3-by-3 with IF, forming three big clusters that are, in their turn, linked also 3-by-3 with IF.

5. Conclusions

Gene expression programming is the most recent development on artificial evolutionary systems and one that brings about a considerable increase in performance due to the crossing of the phenotype threshold [14]. The crossing of the phenotype threshold allows the unconstrained exploration of the search space. Thus, in GEP, the implementation of high-performing search operators such as point mutation, transposition and recombination, is a child's play as any modification made in the genome always results in valid phenotypes or programs.

In this work, the recently invented algorithm was successfully applied to discover Boolean functions on seven-dimensional parameter spaces. The Boolean functions discovered by GEP consist of the solutions to the best known rules at coordinating the behavior of cellular automata in the density-classification task. The understanding of such complex behaviors is only possible if the programs behind the output bits of a CA rule are known. However, for the two rules analyzed here (Coevolution$_1$ and Coevolution$_2$), only the output bits were known. Therefore, the intelligible solutions to the most efficient CA rules at the density-classification task discovered in this work are most valuable for future research in complex emergent behavior.

References

1. Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, MIT Press, 1992.
2. Ferreira, C., 2001. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *Complex Systems*, 13 (2): 87-129.
3. Wolfram, S., *Theory and Applications of Cellular Automata*. World Scientific, 1986.
4. Toffoli, T. and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*. MIT Press, 1987.
5. Gacs, P., G. L. Kurdyumov, and L. A. Levin, 1978. One-dimensional Uniform Arrays that Wash out Finite Islands. *Problemy Peredachi Informatsii* 14, 92-98 (in Russian).
6. Mitchell, M., *An Introduction to Genetic Algorithms*. MIT Press, 1996.
7. Koza, J. R., F. H. Bennett III, D. Andre, and M. A. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, San Francisco, 1999.

8. Juillé, H. and J. B. Pollack. Coevolving the “Ideal” Trainer: Application to the Discovery of Cellular Automata Rules. In J. R. Koza, W. Banzhaf, K. Chellapilla, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, eds., *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Morgan Kaufmann, San Francisco, 1998.
9. Holland, J. H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975 (second edition: MIT Press, 1992).
10. Cramer, N. L., A Representation for the Adaptive Generation of Simple Sequential Programs. In J. J. Grefenstette, ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Erlbaum, 1985.
11. Mitchell, M., J. P. Crutchfield, and P. T. Hraber, 1994. Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments. *Physica D* 75, 361-391.
12. Mitchell, M., P. T. Hraber, and J. P. Crutchfield, 1993. Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations. *Complex Systems* 7, 89-130.
13. Das, R., M. Mitchell, and J. P. Crutchfield, A Genetic Algorithm Discovers Particle-based Computation in Cellular Automata. In Y. Davidor, H.-P. Schwefel, and R. Männer, eds., *Parallel Problem Solving from Nature - PPSN III*, Springer-Verlag, 1994.
14. Dawkins, R., *River out of Eden*. Weidenfeld and Nicolson, 1995.