# Designing Neural Networks Using Gene Expression Programming

**Cândida Ferreira**
Gepsoft, 73 Elmtree Drive
Bristol BS13 8NA
United Kingdom
candidaf@gepsoft.com
http://www.gepsoft.com

*9th Online World Conference on Soft Computing in Industrial Applications, September 20 - October 8, 2004*

An artificial neural network with all its elements is a rather complex structure, not easily constructed and/or trained to perform a particular task. Consequently, several researchers used Genetic Algorithms to evolve partial aspects of neural networks, such as the weights, the thresholds, and the network architecture. Indeed, over the last decade many systems have been developed that perform total network induction. In this work it is shown how the chromosomes of Gene Expression Programming can be modified so that a complete neural network, including the architecture, the weights and thresholds, could be totally encoded in a linear chromosome. It is also shown how this chromosomal organization allows the training/adaptation of the network using the evolutionary mechanisms of selection and modification, thus providing an approach to the automatic design of neural networks. The workings and performance of this new algorithm are tested on the 6-multiplexer and on the classical exclusive-or problems.

## 1. Introduction

An artificial neural network is a computational device that consists of many simple connected units (neurons) that work in parallel. The connections between the units or nodes are weighted usually by real-valued weights. Weights are the primary means of learning in neural networks, and a learning algorithm is used to adjust the weights (e.g., Anderson 1995).

More specifically, a neural network has three different classes of units: input, hidden, and output units. An activation pattern is presented on its input units and spreads in a forward direction from the input units through one or more layers of hidden units to the output units. The activation coming into a unit from other units is multiplied by the weights on the links over which it spreads. All incoming activation is then added together and the unit becomes activated only if the incoming result is above the unit's threshold.

In summary, the basic elements of a neural network are the units, the connections between units, the weights, and the thresholds. And these are the elements that must be encoded in a linear chromosome so that populations of such structures can adapt in a particular selection environment in order to evolve solutions to different problems.

Over the last decade many systems have been developed that evolve both the topology and the parametric values of a neural network (Angeline *et al*. 1993; Braun and Weisbrod 1993; Dasgupta and McGregor 1992; Gruau *et al*. 1996; Koza and Rice 1991; Lee and Kim 1996; Mandischer 1993; Maniezzo 1994; Opitz and Shavlik 1997; Pujol and Poli 1998; Yao and Liu 1996; Zhang and Muhlenbein 1993). The present work introduces a new algorithm, GEP-NN, based on Gene Expression Programming (GEP) (Ferreira 2001) that performs total network induction using linear chromosomes of fixed length (the genotype) that map into complex neural networks of different sizes and shapes (the phenotype). The problems chosen to show the workings of this new algorithm include two problems of logic synthesis: the exclusive-or and the 6-multiplexer.

## 2. Genes with multiple domains for designing neural networks

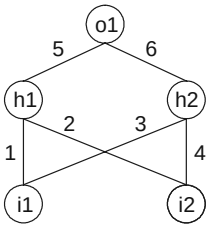
The total induction of neural networks (NN) using GEP, requires further modification of the structural organization developed to manipulate numerical constants (Ferreira 2001, 2003). The network architecture is encoded in the familiar structure of head and tail. The head contains special functions that activate the units and terminals that represent the input units. The tail contains obviously only terminals. Besides the head and the tail, these genes (neural net genes or NN-genes) contain two additional domains, Dw and Dt, encoding, respectively, the weights and the thresholds. Structurally, the Dw comes after the tail and has a length $d_w$ equal to the head

length $h$ multiplied by maximum arity $n$, and Dt has a length $d_t$ equal to $h$. Both domains are composed of symbols representing the weights or thresholds of the neural net.

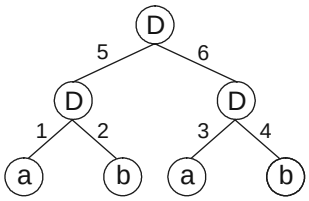
For each NN-gene, the weights and thresholds are created at the beginning of each run, but their circulation is guaranteed by the usual genetic operators of mutation, transposition, and recombination. Nonetheless, a special mutation operator was created that allows the permanent introduction of variation in the set of weights and thresholds.

It is worth emphasizing that the basic genetic operators like mutation or transposition are not affected by Dw and Dt as long as the boundaries of each region are maintained and the alphabets of each domain are not mixed up.

Consider the conventionally represented neural network with two input units ($i_1$ and $i_2$), two hidden units ($h_1$ and $h_2$), and one output unit ($o_1$) (for simplicity, the thresholds are all equal to 1 and are omitted):




It can also be represented as a tree:




where $a$ and $b$ represent, respectively, the inputs $i_1$ and $i_2$ to the network and "D" represents a function with connectivity two. This function multiplies the value of each argument by its respective weight and adds all the incoming activation in order to determine the forwarded output. This output (0 or 1) depends on the threshold which, for simplicity, was set to 1.

We could linearize the above NN-tree as follows:

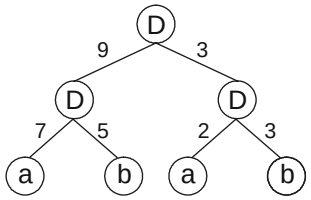
```
0123456789012
DDDabab654321
```

which consists of an NN-gene with the familiar head and tail domains, plus an additional domain Dw for encoding the weights. The values of each weight are kept in an array and are retrieved as necessary. For simplicity, the number represented by the numeral in Dw indicates the order in the array.

Let us now analyze a simple neural network encoding a well-known function, the exclusive-or. Consider, for instance, the chromosome below with $h = 3$ and containing a domain encoding the weights:

```
0123456789012
DDDabab393257
```

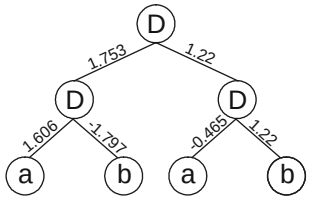
Its translation gives:




For the set of weights:

W = {-1.978, 0.514, -0.465, 1.22, -1.686, -1.797, 0.197, 1.606, 0, 1.753},

the neural network above gives:




(2.1)

which is a perfect solution to the exclusive-or problem.

## 3. Special genetic operators

The evolution of such complex entities composed of different domains and different alphabets requires a special set of genetic operators so that each domain remains intact. The operators of the basic gene expression algorithm (Ferreira 2001) are easily transposed to neural-net encoding chromosomes, and all of them can be used as long as the boundaries of each domain are maintained and alphabets are not mixed up. Mutation was extended to all the domains and continues to be the most important genetic operator. IS and RIS transposition were also implemented in GEP-nets and their action is obviously restricted to heads and tails. However, a special insertion operator was created that operates within Dw and Dt, ensuring the efficient circulation of weights and thresholds in the population (see section 3.1). Another special operator, weights and thresholds' mutation, was also created in order to directly introduce variation in the set of available weights and thresholds (see section 3.3).

The extension of recombination and gene transposition to GEP-nets is straightforward as their actions never result in mixed domains or alphabets. However, for them to work efficiently (i.e., allow an efficient learning), we must be careful in determining which weights and/or thresholds go to which region after the splitting of the chromosomes, otherwise the system is incapable of evolving efficiently. In the case of gene recombination and gene transposition, keeping track of a gene's weights and thresholds is no difficult task, and these operators work very well in GEP-nets. But in one-point and two-point recombination where chromosomes can be split anywhere, it is impossible to keep track of the weights and thresholds. In fact, if applied straightforwardly, these

operators would produce such evolutionary monsters that they would be of little use in multigenic chromosomes. Therefore, for multigenic systems, a special intragenic two-point recombination was created so that the recombination is restricted to a particular gene (see section 3.2).

### *3.1. Domain-specific transposition*

Domain-specific transposition is restricted to the NN-specific domains, Dw and Dt. Its mechanism is, however, similar to IS transposition (Ferreira 2001). This operator randomly chooses the chromosome, the gene with its respective Dw plus Dt (if we use the same symbols to represent the weights and the thresholds, we can treat Dw and Dt as one big domain), the first position of the transposon, the transposon length, and the target site (also chosen within Dw plus Dt).

Consider the chromosome below with $h = 4$ (Dw and Dt are shown in different shades):

```
0123456789012345678901234567890123456
DTQaababaabbaabba05717457362846682867
```
(3.1)

where "T" represents a function of three arguments and "Q" represents a function of four arguments. Suppose that the sequence "46682" was chosen as a transposon and that the insertion site was bond 4 in Dw (between positions 20 and 21). Then the following chromosome is obtained:

```
0123456789012345678901234567890123456
DTQaababaabbaabba05714668274573628466
```
(3.2)

Note that the transposon might be any sequence in Dw or Dt, or even be part Dw and part Dt like in the example above. Note also that the insertion site might be anywhere in Dw or Dt as the symbols used to represent the weights and the thresholds are the same. Remember, however, that the values they represent are different for they are kept in different arrays. Suppose that the arrays below represent the weights and the thresholds of chromosome (3.1) above:

W = {-1.64, -1.834, -0.295, 1.205, -0.807, 0.856, 1.702, -1.026, -0.417, -1.061}
T = {-1.14, 1.177, -1.179, -0.74, 0.393, 1.135, -0.625, 1.643, -0.029, -1.639}

Although the new chromosome (3.2) obtained after transposition has the same topology and uses exactly the same arrays for its expression, a different neural network is encoded in this chromosome (Figure 1). Indeed, with domain-specific transposition the weights and thresholds are moved around and new combinations are tested.

### *3.2. Intragenic two-point recombination*

Intragenic two-point recombination was created in order to allow the modification of a particular gene without interfering with the other sub-neural nets encoded in other genes. The mechanism of this kind of recombination is exactly the same as in two-point recombination, with the difference that the recombination points are chosen within a particular gene (see Figure 2).

Consider the following parent chromosomes composed of two genes, each with a weights domain ($W_{i,j}$ represents the weights of gene $j$ in chromosome $i$):

$W_{0,1}$ = {-0.78, -0.521, -1.224, 1.891, 0.554, 1.237, -0.444, 0.472, 1.012, 0.679}
$W_{0,2}$ = {-1.553, 1.425, -1.606, -0.487, 1.255, -0.253, -1.91, 1.427, -0.103, -1.625}

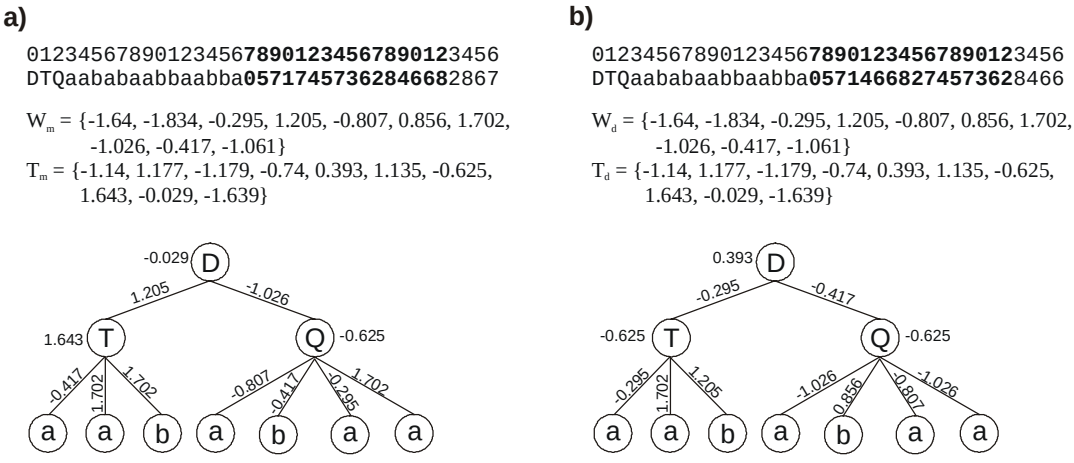



**Figure 1.** Testing new combinations of existing weights and thresholds by domain-specific transposition. **a)** The mother neural network. **b)** The daughter neural network created by domain-specific transposition. Note that the network architecture is the same for both mother and daughter and that $W_m = W_d$ and $T_m = T_d$. However, mother and daughter are different because different combinations of weights and thresholds are expressed in these individuals.

```
012345678901234560123456789012345 6
TTababaab14393255QDbabbabb96369304-[0]
Qaabbbabb97872192QDbabbaaa81327963-[1]
```

$W_{1,1}$ = {-0.148, 1.83, -0.503, -1.786, 0.313, -0.302, 0.768, -0.947, 1.487, 0.075}
$W_{1,2}$ = {-0.256, -0.026, 1.874, 1.488, -0.8, -0.804, 0.039, -0.957, 0.462, 1.677}

Suppose that the first gene was chosen to recombine and point 1 (between positions 0 and 1) and point 12 (between positions 11 and 12) were chosen as recombination points. Then the following offspring is formed:

```
012345678901234560123456789012345 6
Taabbbabb97893255QDbabbabb96369304-[0]
QTababaab14372192QDbabbaaa81327963-[1]
```

with weights encoded in exactly the same arrays as the parents. However, due to recombination, the weights expressed in the parents are different from those expressed in the offspring (compare their expressions in Figure 2).

It is worth emphasizing that this gene-restricted two-point recombination allows a greater control of the recombination effects and, consequently, permits a finer tuning of evolution. If we were to use one-point and two-point recombination as used in the basic gene expression algorithm, i.e., disrupting chromosomes anywhere, the fine adjustment of the weights would be an almost impossible task. Restricting two-point recombination to only one gene, however, ensures that only this gene is modified and, consequently, the weights and thresholds of the remaining genes are kept in place.

Remember, however, that intragenic two-point recombination is not the only source of recombination in multigenic neural nets: gene recombination is fully operational in these systems and it can be combined with gene transposition to propel evolution further. And in unigenic systems, the standard one-point and two-point recombination are also fully operational as only one gene is involved.

### *3.3. Direct mutation of weights and thresholds*

In the previous sub-sections it was shown that all genetic operators contribute directly or indirectly to move the weights and thresholds around. And, in fact, this constant shuffling

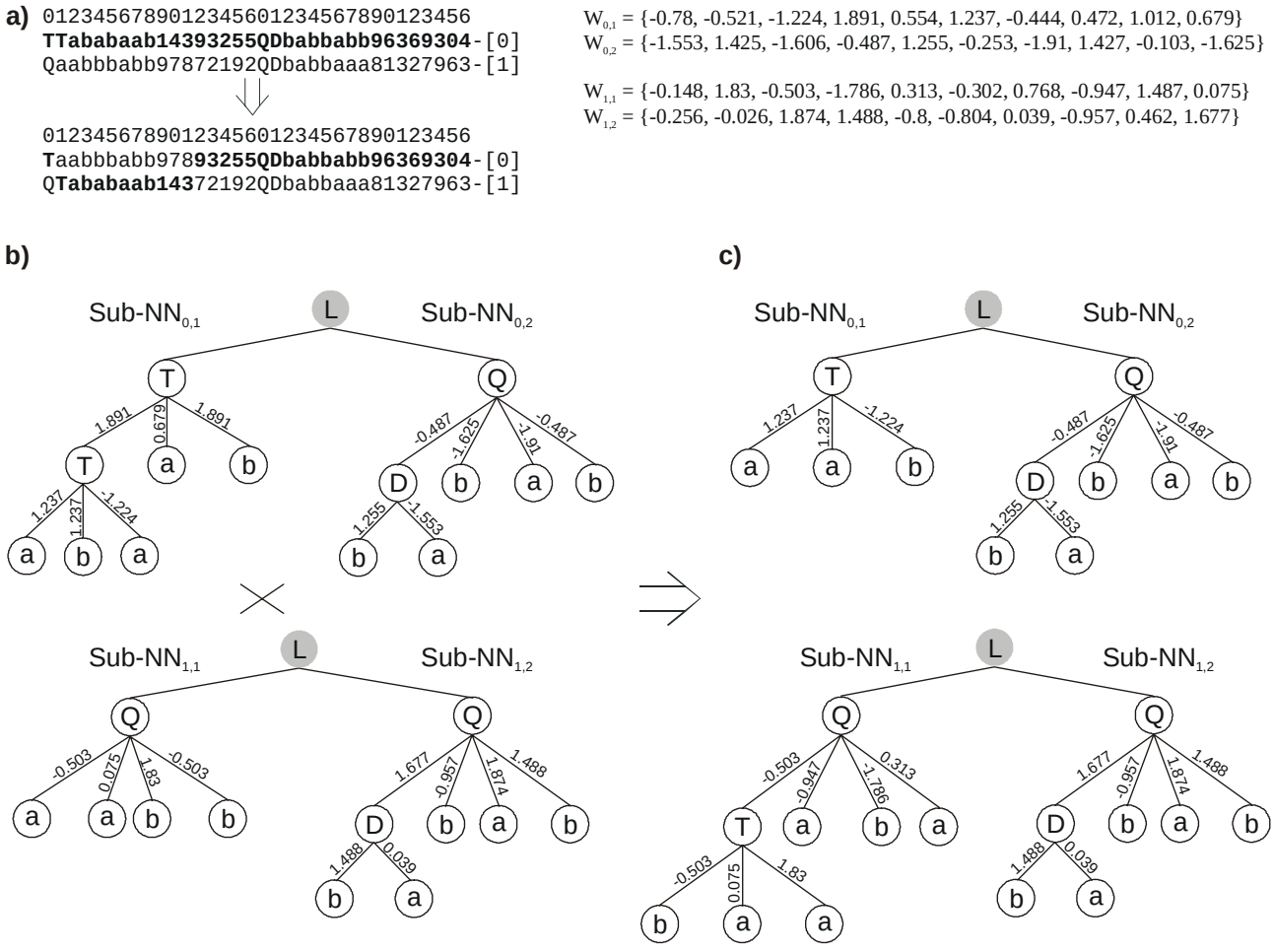



**Figure 2.** Intragenic two-point recombination in multigenic chromosomes encoding neural networks. **a)** An event of intragenic two-point recombination between two parent chromosomes resulting in two new daughter chromosomes. Note that the set of weights is not modified by recombination. **b)** The sub-NNs codified by the parent chromosomes. **c)** The sub-NNs codified by the daughter chromosomes. Note that the sub-NNs encoded in the second gene are not modified. "L" represents a generic linking function.

of weights and thresholds is more than sufficient to allow an efficient evolution of GEP-nets as long as an appropriate number of weights and thresholds is randomly created at the beginning of each run. However, special mutation operators that replace the value of a particular weight or threshold by another can also be easily implement (see Figure 3).

This operator randomly selects particular targets in the arrays in which the weights or thresholds are kept, and randomly generates a new real-valued number. Consider for instance the array:

$W_{i,j}$ = {-0.433, -1.823, 1.255, 0.028, -1.755, -0.036, -0.128, **-1.163**, 1.806, 0.083}

encoding the weights of gene *j* on chromosome *i*. Suppose a mutation occurred at position 7, changing the weight -1.163 occupying that position into -0.494, obtaining:

$W_{i,j}$ = {-0.433, -1.823, 1.255, 0.028, -1.755, -0.036, -0.128, **-0.494**, 1.806, 0.083}

The consequences of this kind of mutation are very diverse: they might be neutral in effect (for instance, when the gene itself is neutral or when the weight/threshold has no expression on the sub-neural net) or they might have manifold effects (for instance, if the weight/threshold modified happened to be used more than once in the expression of the sub-NN as shown in Figure 3).

Interestingly, this kind of mutation seems to have a very limited importance and better results are obtained when this operator is switched off. Indeed, the direct mutation of numerical constants in function finding problems produces identical results (Ferreira 2003). Therefore, we can conclude that a well dimensioned initial diversity of constants, be they numerical constants of a mathematical expression or weights/thresholds of a neural net, is more than sufficient to allow their evolutionary tuning. In all the problems presented in this work, a set of 10 weights W = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} was used.

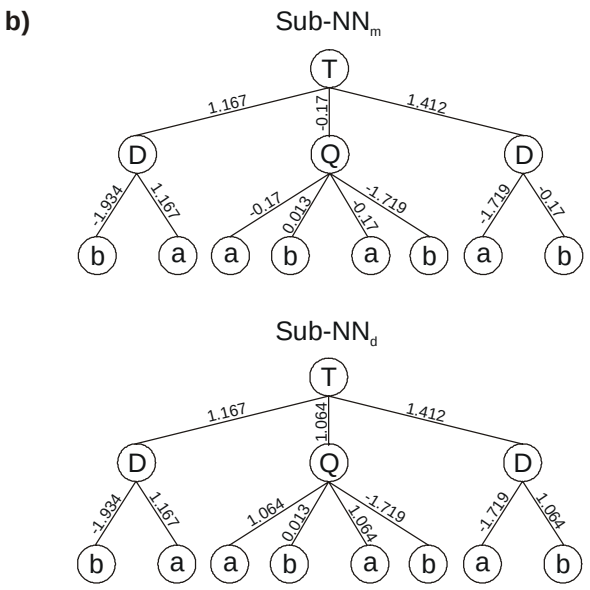



**Figure 3.** Illustration of direct mutation of weights. **a)** The mother and daughter chromosomes with their respective weights. In this case, weights at positions 0 and 2 were mutated. Note that the mother and daughter chromosomes are equal. **b)** The mother and daughter neural nets encoded in the chromosomes. Note that the point mutation at position 2 (-0.17) has manifold effects as this weight appears four times in the neural network. Note also that the mutation at position 0 is an example of a neutral mutation as it has no expression on the neural net (indeed, mutations at positions 4, 6, and 9 would also be neutral).

## 4. Solving problems with neural networks designed by gene expression programming

The problems chosen to illustrate the evolution of linearly encoded neural networks are two well-known problems of logic synthesis. The first, the exclusive-or problem, was chosen both for its historical importance in the neural network field and for its simplicity, allowing an easy understanding of the evolved neural net. The second, the 6-bit multiplexer, is a rather complex problem and can be useful for evaluating the efficiency of this new algorithm.

### *4.1. Neural network for the exclusive-or problem*

The XOR is a simple Boolean function of two activities and, therefore, can be easily solved using linearly encoded neural networks. Its rule table is shown in Table 1.

The functions used to solve this problem have connectivities 2, 3, and 4, and are represented, respectively, by "D", "T", and "Q", thus the function set F = {D, T, Q}; the terminal set T = {a, b}; and the set of weights W = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} with values randomly chosen from the interval [-2, 2]. For the experiment summarized in the first column of Table 2, an *h* = 4 was chosen and, therefore, hundreds of different correct solutions to the XOR function were found. Most of them are more complicated than the conventional solution (2.1) shown above which uses seven nodes; others have the same degree of complexity evaluated in terms of total nodes; but surprisingly others are more parsimonious than the mentioned conventional solution for the XOR function as will next be shown.

The first solution found in run 0 of the experiment summarized in the first column of Table 2 is shown below:

**Table 1**
Lookup table for the exclusive-or function.

| a | b | o |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Table 2**
Parameters for the exclusive-or problem.

| | Redundant System | Compact System |
|---|---|---|
| Number of runs | 100 | 100 |
| Number of generations | 50 | 50 |
| Population size | 30 | 30 |
| Number of fitness cases | 4 | 4 |
| Function set | D T Q | D T Q |
| Terminal set | a b | a b |
| Weights array length | 10 | 10 |
| Weights range | [-2, 2] | [-2, 2] |
| Head length | 4 | 2 |
| Number of genes | 1 | 1 |
| Chromosome length | 33 | 17 |
| Mutation rate | 0.061 | 0.118 |
| One-point recombination rate | 0.7 | 0.7 |
| IS transposition rate | 0.1 | -- |
| IS elements length | 1 | -- |
| RIS transposition rate | 0.1 | -- |
| RIS elements length | 1 | -- |
| Dw-specific transposition rate | 0.1 | 0.1 |
| Dw-specific IS elements length | 2,3,5 | 2,3,5 |
| Success rate | 77% | 30% |

```
012345678901234567890123456789012
TQaTaaababbbabaaa6085977238275036
```

W = {1.175, 0.315, -0.738, 1.694, -1.215, 1.956, -0.342, 1.088, -1.694, 1.288}

Its expression is shown in Figure 4. It is a rather complicated solution to the XOR function, but remember that evolutionary algorithms thrive in slightly redundant architectures (Ferreira 2002) and, as shown in Table 2, the success rate for this problem using this non-compact chromosomal organization is higher (77%) than the obtained with more compact organizations with $h = 2$ (30%).

However, GEP can be useful to search for parsimonious solutions, and a very interesting parsimonious solution to the XOR function was found in another experiment. The parameters used per run in this experiment are summarized on the second column of Table 2. Note that the compact organization with $h = 2$ was chosen in order to search for solutions more parsimonious than the canonical solution to the XOR function. One such solution is shown below:

```
01234567890123456
TDbabaabb88399837
```

W = {0.713, -0.774, -0.221, 0.773, -0.789, 1.792, -1.77, 0.443, -1.924, 1.161}

which is a perfect, extremely parsimonious solution to the XOR problem. Its full expression is shown in Figure 5. Indeed, several perfect solutions with this kind of structure were found in this experiment.

**a)**
012345678901234567890123456789012
TQaTaaababbbabaaa6085977238275036
W = {1.175, 0.315, -0.738, 1.694, -1.215, 1.956, -0.342, 1.088, -1.694, 1.288}

**b)**

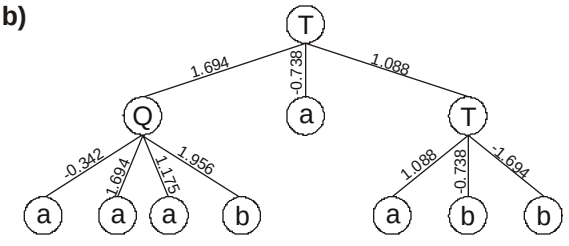



**Figure 4.** A perfect, slightly complicated solution to the exclusive-or problem evolved with GEP neural networks. **a)** Its chromosome and respective weights. **b)** The fully expressed neural network encoded in the chromosome.

**a)**
01234567890123456
TDbabaabb88399837
W = {0.713, -0.774, -0.221, 0.773, -0.789, 1.792, -1.77, 0.443, -1.924, 1.161}

**b)**

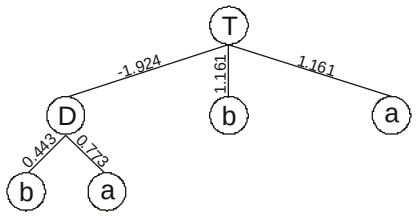



**Figure 5.** A perfect, extremely parsimonious solution to the exclusive-or problem discovered with GEP designed neural networks. **a)** Its chromosome and corresponding array of weights. **b)** The fully expressed neural network encoded in the chromosome.

### *4.2. Neural network for the 6-multiplexer*

The 6-bit multiplexer is a complex Boolean function of six activities. Its rule table is shown in Table 3.

**Table 3**
Lookup table for the 6-multiplexer. The output bits are given in lexicographic order starting with 000000 and finishing with 111111.

```
00000000 11111111 00001111 00001111
00110011 00110011 01010101 01010101
```

In order to simplify the analysis, a rather compact chromosomal organization was chosen and the "Q" function was not included in the function set. Thus, for this problem, F = {3U, 3D, 3T}, where "U" represents a function with connectivity one; T = {a, b, c, d, e, f}, representing the six arguments to the 6-multiplexer function; and W = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, each taking values from the interval [-2, 2].

For the experiment summarized in the first column of Table 4, single-gene chromosomes were chosen so that the simulation of the 6-multiplexer function, a four modular function, went totally unbiased. One of the most parsimonious solutions designed is shown in Figure 6.

Obviously, we could explore the multigenic nature of GEP chromosomes and evolve multigenic neural networks. The solutions found are, however, structurally more constrained as we have to choose some kind of linking function (Ferreira 2001) to link the sub-neural nets encoded by each gene. For this problem, the Boolean function OR was chosen to link the sub-NNs. (If the mixing of OR with "U", "D", and "T" functions is confusing, think of OR as a function with connectivity two with a threshold and weights all equal to 1, and you have a neural net for the OR function.)

In the experiment summarized in the second column of Table 4, four genes posttranslationally linked by OR were used. The first solution found in this experiment is shown in Figure 7. Note that some weights in genes 1 and 2 have identical values, and that the same happens for genes 3 and 4. This most probably means that these genes share a common ancestor.

## 5. Conclusions

The new algorithm presented in this work allows the complete induction of neural networks encoded in linear chromosomes of fixed length (the genotype) which, nonetheless, allow the evolution of neural networks of different sizes and shapes (the phenotype). Both the chromosomal organization and the genetic operators especially developed to evolve neural networks allow an unconstrained search throughout the solution space as any modification made in the genotype always results in valid phenotypes. Furthermore, as shown for the 6-multiplexer problem presented in this work, the multigenic nature of GEP-nets can be further explored to evolve complex neural networks with multiple outputs.

**Table 4**
Parameters for the 6-multiplexer problem.

| | Unigenic System | Multigenic System |
|---|---|---|
| Number of runs | 100 | 100 |
| Number of generations | 2000 | 2000 |
| Population size | 50 | 50 |
| Number of fitness cases | 64 (Table 3) | 64 (Table 3) |
| Function set | 3U 3D 3T | 3U 3D 3T |
| Terminal set | a b c d d e f | a b c d d e f |
| Linking function | -- | O |
| Weights array length | 10 | 10 |
| Weights range | [-2, 2] | [-2, 2] |
| Head length | 17 | 5 |
| Number of genes | 1 | 4 |
| Chromosome length | 103 | 124 |
| Mutation rate | 0.044 | 0.044 |
| Intragenic two-point recombination rate | 0.6 | 0.6 |
| Gene recombination rate | -- | 0.1 |
| Gene transposition rate | -- | 0.1 |
| IS transposition rate | 0.1 | 0.1 |
| IS elements length | 1,2,3 | 1,2,3 |
| RIS transposition rate | 0.1 | 0.1 |
| RIS elements length | 1,2,3 | 1,2,3 |
| Weights mutation rate | 0.002 | 0.002 |
| Dw-specific transposition rate | 0.1 | 0.1 |
| Dw-specific IS elements length | 2,3,5 | 2,3,5 |
| Success rate | 4% | 6% |

**a)** `TbDTTTfTTaUDcUUTTafeefebabbdabffddfcfeeeabcabfabdcfe...`
`...709761631479459597193997465381760511137453583952159`
W = {0.241, 1.432, 1.705, -1.95, 1.19, 1.344, 0.925, -0.163, -1.531, 1.423}

**b)**

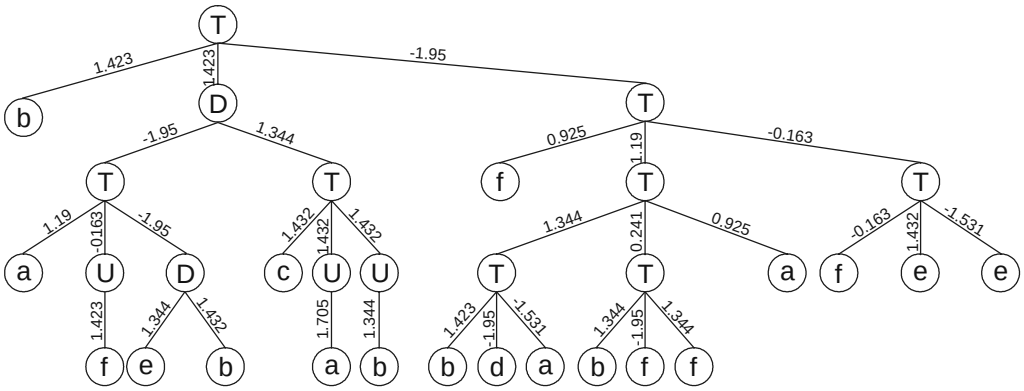


**Figure 6.** A perfect solution to the 6-multiplexer function discovered with GEP designed neural networks. **a)** Its chromosome and corresponding array of weights. **b)** The fully expressed neural network encoded in the chromosome.

**a)**
```
TecTDdfafabdddfa487674791701403
TDcbTbadddfceacc501702156029560
TfTTUbadbcdffdce593993321226318
TDTbaceaaeeacacd072636270049968
```

$W_1$ = {1.126, 0.042, 1.588, -0.03, -1.91, 1.83, -0.412, 0.607, -0.294, -0.659}
$W_2$ = {-1.961, 1.161, 1.588, -0.03, -1.91, 1.762, -0.412, -0.121, -0.294, -0.659}
$W_3$ = {1.558, -0.69, 0.921, 0.134, 0.468, -1.534, 0.966, 1.399, 0.023, 0.915}
$W_4$ = {1.558, 0.767, 0.076, 0.071, 0.468, -1.534, 1.387, -1.857, -1.88, 0.331}

**b)**

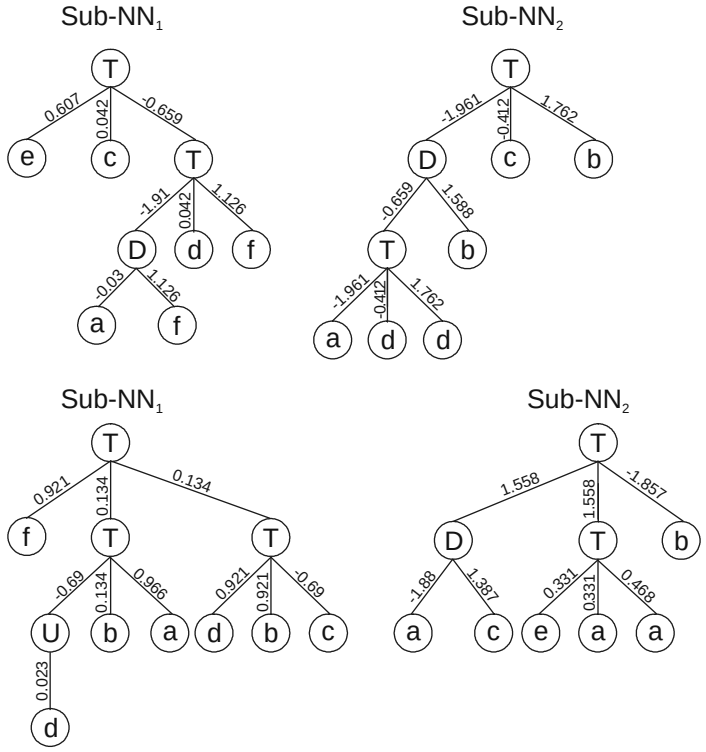



**Figure 7.** A perfect solution to the 6-multiplexer problem encoded in a four-genic chromosome. **a)** Its chromosome with each gene shown separately. $W_1$-$W_4$ are the arrays containing the weights of each gene. **b)** The sub-neural networks codified by each gene. In this perfect solution, the sub-neural nets are linked by OR.

## Bibliography


Anderson, J. A., *An Introduction to Neural Networks*, MIT Press, 1995.

Angeline, P. J., G. M. Saunders, and J. B. Pollack, 1993. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5: 54-65.

Braun, H. and J. Weisbrod. Evolving feedforward neural networks. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*. Innsbruck, Springer-Verlag, 1993.

Dasgupta, D. and D. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the International Conference on Combinations of Genetic Algorithms and Artificial Neural Networks*, pp. 87-96, 1992.

Ferreira, C., 2001. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13 (2): 87-129.

Ferreira, C., 2002. Genetic representation and genetic neutrality in gene expression programming. *Advances in Complex Systems*, 5 (4): 389-408.

Ferreira, C. Function finding and the creation of numerical constants in gene expression programming. In J. M. Benitez, O. Cordon, F. Hoffmann, and R. Roy, eds., *Advances in Soft Computing: Engineering Design and Manufacturing*, pp. 257-266, Springer-Verlag, 2003.

Gruau, F., D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 81-89, Cambridge, MA, MIT Press, 1996.

Koza, J. R. and J. P. Rice. Genetic generation of both the weights and architecture for a neural network. In *Proceedings of the International Joint Conference on Neural Networks*, Volume II, IEEE Press, 1991.

Lee, C.-H. and J.-H. Kim. Evolutionary ordered neural network with a linked-list encoding scheme. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pp. 665-669, 1996.

Mandischer, M. Representation and evolution of neural networks. In R. F. Albrecht, C. R. Reeves, and U. C. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*, pp. 643-649, Springer Verlag, 1993.

Maniezzo, V., 1994. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5 (1): 39-53.

Opitz, D. W. and J. W. Shavlik, 1997. Connectionist theory refinement: Genetically searching the space of network topologies. *Journal of Artificial Intelligence Research*, 6: 177-209.

Pujol, J. C. F. and R. Poli, 1998. Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence Journal, Special Issue on Evolutionary Learning*, 8(1): 73-84.

Yao, X. and Y. Liu, 1996. Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, 91(1): 83-90.

Zhang, B.-T. and H. Muhlenbein, 1993. Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex Systems*, 7: 199-220.