CIMCA 2004 Proceedings/ISBN :1740881885 M. Mohammadian (Ed.)
12-14 July 2004, Gold Coast - Australia

A Permutation Problem Free Modified Cellular Encoding for Evolving Artificial Neural Networks

Mohammad Masud Hasan[1], Md. Monirul Islam[2] and K. Murase[3]
[1,2]Department of Computer Science and Engineering, BUET, Dhaka 1000
[3]Dept. of Human and Artificial Intelligent Systems, Univ. of Fukui, Japan
Emails: [1]hasanmm@cse.buet.ac.bd, [2]mdmonirul@yahoo.com, [3]murase@synapse.his.fukui-u.ac.jp

Abstract

This paper presents a new indirect encoding scheme, modified cellular encoding (MCE), for evolving artificial neural networks (ANNs). The cellular encoding (CE) method is modified in such a way that it does not suffer from the well-known permutation problem or competing conventions problem of genetic algorithms for evolving ANNs. The functionality of some program symbols in CE is changed; new rules are added. As a consequence, MCE is able to apply crossover operator without facing permutation problem in the genetic search. It is shown in this paper that addition / deletion of nodes / connections can evidently be done by crossover alone. Attempts are also taken to minimize behavioral disruption between parents and their offspring.

Key words: Artificial neural network, Indirect encoding, Direct encoding, Evolutionary system, Evolution of behaviors, Genotype, Phenotype, Crossover, Training.

1 INTRODUCTION

Most applications of artificial neural networks (ANNs) use feedforward networks and variants of the classical backpropagation (BP) algorithm. All these training algorithms assume a fixed ANN architecture. They only train weights in the fixed architecture that includes both connectivity and node transfer functions. Many attempts have also been made in designing ANN architectures automatically, such as various constructive and pruning algorithms [8, 9, 10, 11, 12]. Angeline *et.al.* [13] indicate two problems of such hill climbing methods: they may be trapped at local optima and they do not investigate complete class of network architectures. That is why researchers [2, 7] argued on behalf of evolutionary algorithms for finding a near optimal system in the ANN architecture search space.

The central task in evolving ANNs is finding a genetic representation, or encoding, for an ANN [7]. It dictates how the search landscape is structured, and how scalable the method is. There are currently maybe dozens of different encoding methods [5]. Most of the ANN implementations appeared in direct encoding forms. But the detrimental effect of the permutation problem in such encoding scheme makes the evolutionary system for ANNs inefficient. Xin Yao *et. al.* [2] did not use crossover operator in their EPNet algorithm, since it is not clear what building blocks actually are in this situation. Moreover, “the large amount of human effort, intervention, and experience required to find the proper architecture for direct encoding make it less attractive” [6]. Hence, Gruau [1] proposed an initial step towards the development of a systematic method for the automatic exploration of ANN search space. He defines an indirect encoding scheme named cellular encoding (CE) capable of generating a wide variety of ANNs from a small class of operators. But the permutation problem still presents in his approach.

This paper describes a new permutation problem free indirect encoding scheme for evolving ANNs. We choose a complete and close set of program symbols from CE [1]. We modify the functionality of some symbols in such a way that this upgradation would lead to the permutation problem free genotypes of ANNs. As a result, the difficulties in producing highly

fit offspring would no longer exist with crossover operators. It is argued in this paper that the crossover operations with the symbols of our modified CE technique suffice to generate all kinds of architectural evolutions like deletion / addition of nodes / connections of the ANNs. This implies the reduction of different user specified parameters in the evolutionary system.

The organization of the rest of this paper is as follows: Section II represents the permutation problem and properties of a new cellular encoding technique to avoid this problem; Section III discusses about the crossover operator in the genetic evolution system; Section IV concludes with a summary of the paper and a few remarks.

2 MODIFIED CELLULAR ENCODING & PERMUTATION PROBLEM

We will discuss first the program symbols with which we work with and the reason of our choice. We will define permutation problem and show how this problem is absent in our approach.

A. Cellular Encoding (CE)

CE is a method for encoding families of similarly structured ANNs [1, 3]. The cellular code is represented as a grammar tree (called program symbol tree or PST) with ordered branches whose nodes are labeled with name of program symbols. A particular ANN is specified through the breadth first search traversal of this PST and application of a sequence of graph transformations to an initial graph. The transformations operate upon graphs which may have two classes of nodes, either cells or neurons. These two nodes differ in that each cell has associated with it a PST that specifies how that cell will ultimately be replaced by neurons. Neurons are the nodes, which make up the ANN that ultimately results from the process. In the end, all cells must have been transformed into neurons.

To summarize visually, Figure 1(a) shows a simple cellular instance with the starting graph consisting of a single cell *a*, whose reading head r_a indicates a starting execution at the root of its PST. In this case, as is required to obtain ultimately a functional ANN, the single cell in the starting graph has initial connections to inputs and outputs. After execution of program symbol SEQ, which divides the cell sequentially, the new cellular instance has two cells whose reading heads are r_a and r_b as shown in Figure 1(b).

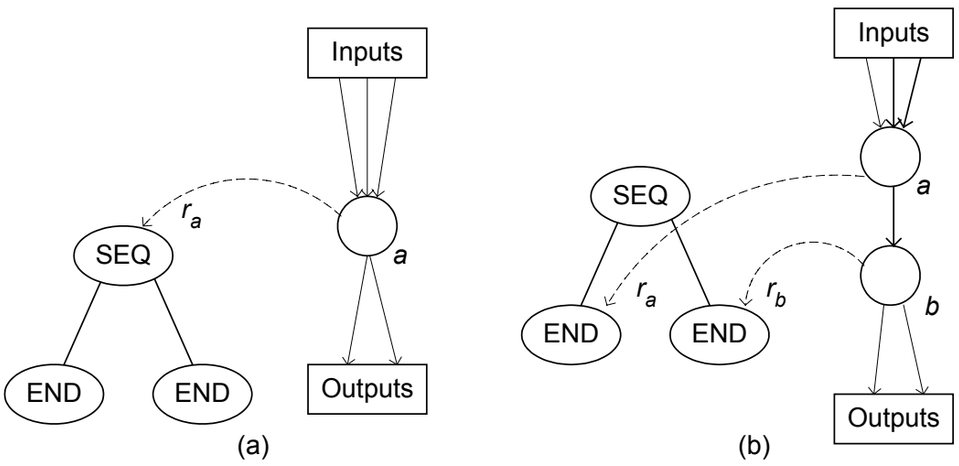

Fig. 1. An example of the execution of PST. (a) Before execution. (b) After realization of the first symbol.

B. The Program Symbol Set

An encoding scheme is complete with respect to the set of architectures (resp. ANNs) if any element of the set can be encoded [1]. And an architecture encoding scheme is closed with respect to a set of architectures, if every possible code develops an architecture in this set.

We initially choose program symbol set {ACYC, END, PAR, SEQ, CUT, WAIT, INCLR, MRG} which is closed and complete. There are other sets of symbols that also hold such features, but this one is simpler and minimum in cardinality [1]. Among the above program symbols, link register is used only for CUT and MRG symbols. It means that INCLR (and DECLR, [3]) is used with CUT and MRG symbols. If we use parameters with these two symbols to determine the link to be processed we do not need INCLR any more.

Again, the symbol WAIT is a skip operation. It is introduced since in some cases the order of cell execution should be varied to be able to represent all ANNs, i.e. for more compact codes. As mentioned in [3], the benefits and drawbacks of WAIT were not analyzed. CE needs WAIT because of Gruau's implementation of connections. He implements them as a matrix, and when a node is added, the matrix is resized to include a new row and new column, and Gruau is very unclear as to how these rows and columns are initialized. Moreover, while we attempt to get rid of permutation problem form the encoding scheme, we see that WAIT enables operations, which are independent of it, to be reordered. The possibilities of shuffling program symbols with WAIT in a PST permit different encodings for the same neural network. That is why we do not use WAIT directly. We use the functionality of WAIT implicitly whenever needed. For example, a MRG operation that causes deletion of a cell will be delayed until the reading head of the cell is END. Such implicit use of WAIT's functionality not only make sure to remain the completeness and closure properties defined by Gruau but also purge the permutation problem associated with WAIT symbol.

Hence the five program symbols we use for CE without recurrent link are END, PAR, SEQ, CUT, and MRG. This modification of CE scheme does not weigh down compactness property; the closure & completeness properties also hold as well.

C. Addressing Permutation Problem

We want to resolve permutation problem (also called competing conventions problem [4] or structural-functional mapping problem [20]) in the encodings of ANNs. It is caused by the many to one mapping from genotypes to phenotypes. In general, it is happened when any permutation of the hidden nodes produces behaviorally equivalent ANN but with different genotypic representation. This problem not only makes the evolution inefficient, but also makes crossover operators more difficult to produce quality descendants [2, 5].

D. Resolving Permutation Problem

To solve the permutation problem we modify some properties of the program symbols of CE. To understand clearly, let us observe the program symbols more details. Permutation problem arises whenever different orientations of nodes in hidden layers are possible for behaviorally same ANN. Two children created by program symbol SEQ do not have this property since here the output of one child is the input to another child. This link will totally reverse if we arrange these two children in another orientation while creation and hence will get ANN of different phenotype. Program symbol END has no arity and CUT, MRG have single arity [3]. Therefore permutation problem cannot be generated from their children. But the program symbol PAR is of double arity and its two children grow in parallel having same input-output links. If we arrange the two children in different order we will have different genotypic representations for the same phenotype.

This problem arises from the poor definition of the execution of PAR symbol while interpreting PST (i.e. CE of ANN). According to [1], after execution of PAR the reading head of the first child cell will be placed on left subtree and the reading head of the second child cell will be placed on right subtree. If we place the reading head of the first child cell on right subtree and the reading head of the second child cell on left subtree (which is another CE), we will have different ANN of same behavior.

We redefine the interpretation of PAR symbol so that the decision of placing reading head of a child, either at the left subtree or at the right subtree, will be taken randomly at the time of execution of PAR symbol while interpreting the PST (addressed by Property 3 given below). This ensures that the two children of PAR have equal probability to place reading head at the left (or right) subtree. Consequently, in the PST, any order of two subtrees (hence constructing different redundant CEs) below PAR symbol can produce same set of behaviorally equivalent ANNs. If we allow both orders, we will have many to one mapping from genotypes to phenotypes. Instead, we permit to appear the program symbols under the PAR symbol in the PST according to a fixed order only (addressed by Property 1 given below). Any order here may suffice (Property 3 explains why it is true), we can select program symbols randomly from the order given in Property 1. This restriction just removes redundancy in the search space by not allowing the occurrences of multiple CEs for the same phenotype.

***Property* 1:** The two children (program symbols) of the PAR symbol in PST can appear according to the following order only, given in descending priority, PAR, SEQ, CUT, MRG, END.

When the two children of PAR in PST are of same type symbol, there is no problem until they are both PAR again. For the case of two PAR children of a parent PAR, we formulate another restriction.

***Property* 2:** At most one child of a PAR symbol in the PST can be a PAR symbol again. According to Property 1, it should be the left child if there is one.

In fact, Property 3 enables us to represent a PAR symbol with two PAR children by a series of PAR symbols in PST. Hence Property 2 does not restrict search space at all; rather it removes redundancy from the search space by stopping to encode same phenotypic ANN in different ways.

Before going to Property 3, we first introduce a new term Random Selection Group (RSG). If a PAR symbol in the PST has two non-PAR symbols (symbols other than PAR), they (including their corresponding subtrees) constitute the RSG of that PAR symbol. If one child of the PAR symbol is PAR again (it should be left child by Property 1), then the two non-PAR children (along with corresponding subtrees) of this child PAR symbol and the right child (along with corresponding subtree) of the parent PAR symbol constitute the RSG for that parent PAR symbol. This definition of RSG goes recursively.

***Property* 3:** During the interpretations of the children of a particular PAR symbol in PST, we can choose randomly and exclusively any symbol (including corresponding subtree) from the RSG of that particular PAR symbol for all the positions of non-PAR symbols children.

Here exclusively means that if we choose one element from RSG for a position we will exclude this form the RSG and not consider any more. All elements have equal probability to be chosen.

An important point to note is while representing CE with RSGs, we will keep symbols in a RSG according to the order given in Property 1. That is, highest priority symbol of a RSG will

appear in the highest level of positions that constitute the RSG. If multiple CUT (or MRG) symbols appear, they also are ordered based on the parameters.

Figure 2 describes the interpretations of Properties 1 and 3. ANNs shown in Figure 2(a) and 2(b) are of same phenotype [2]. CE shown in Figure 2(c) can generate both ANNs. Here two children of PAR, CUT 1 (along with its subtree) and END constitute RSG of the parent PAR. According to Property 3, while interpreting the PST we can choose for the left child of PAR anyone from RSG. If we choose CUT 1 for the left child execution we get ANN 2(a). If we choose END for the left child execution we get ANN 2(b). CE shown in Figure 2(d) is invalid. According to Property 1 CUT has higher priority than END, hence CUT should be left child and END should be the right child of PAR as shown in Figure 2(c).

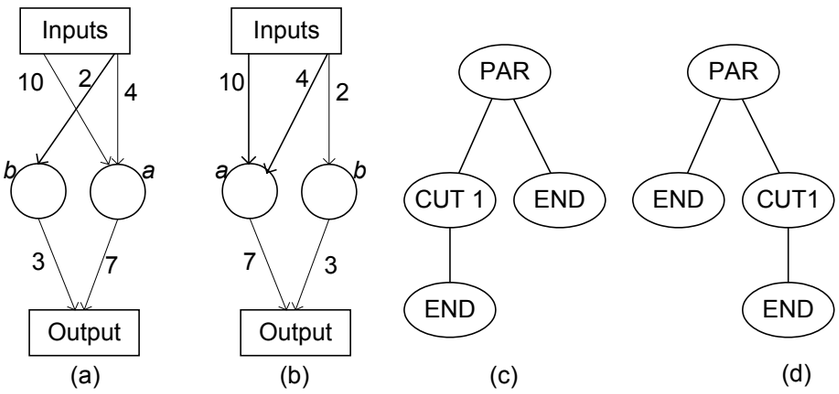

Fig. 2. An example describing Properties 1 and 3. (a), (b) ANNs of same phenotype. (c) CE of the architecture of both (a) and (b). (d) an invalid encoding of the ANNs.

Figure 3 explains Properties 2 and 3. Figure 3(a) is a ANN that can be encoded in several ways. Among these only the CE shown in 3(b) is valid according to Property 2. Figure 3(c) is an invalid representation since it has two PAR children of the parent PAR symbol. Any permutation of the four nodes ($P(4, 4) = 4! = 24$ ways) of the ANN in Figure 3(a) will produce behaviorally same ANNs. Figure 3(b) can generate any of these ANNs. The RSG of the root PAR symbol constitute of CUT 1, CUT 3, along with their corresponding subtrees, and the two ENDs of the last PAR symbols. We have 4 ways to select from RSG for the first right child. For the second right child we have the remaining 3 ways. For the other two children we have 2 and 1 ways. So, total ways to execute the PST is $4 \times 3 \times 2 \times 1 = 24$. The general proof is shown in the following Proposition / Theorem.

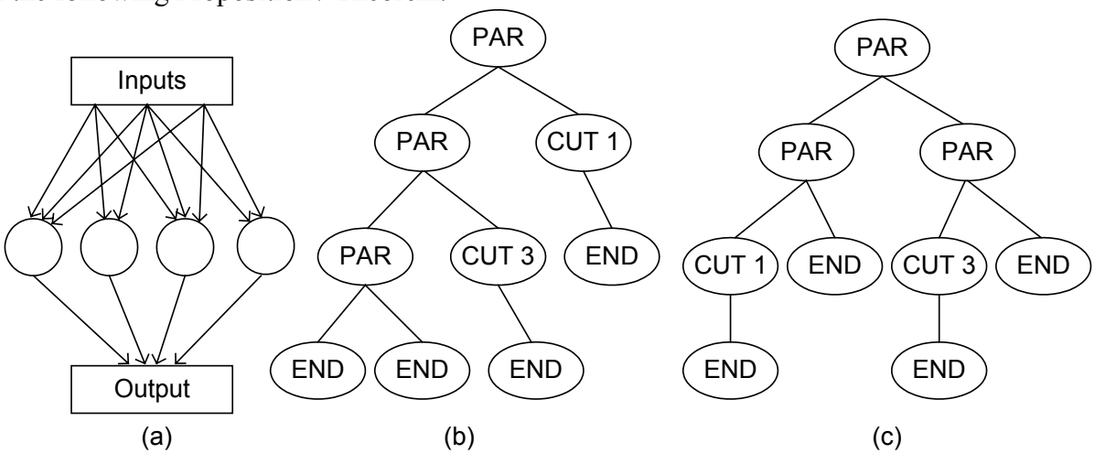

Fig. 3. An example describing Properties 2 and 3. (a) a given ANN. (b) its valid CE. (c) an invalid encoding.

***Proposition* 1:** Property 1, Property 2, and Property 3 allow a unique CE for all ANNs of same phenotype, thus removing permutation problem.

***Proof*:** Since program symbols SEQ, CUT, MRG and END do not contribute in the permutation problem; we concentrate on PAR symbol only. As a result of our modification of the behaviors of the original PAR symbol we will get the only CE shown in Figure 4(b) for the ANN as in Figure 4(a). No other organization of PAR symbols to produce this ANN is valid by our Properties 1 and 2.

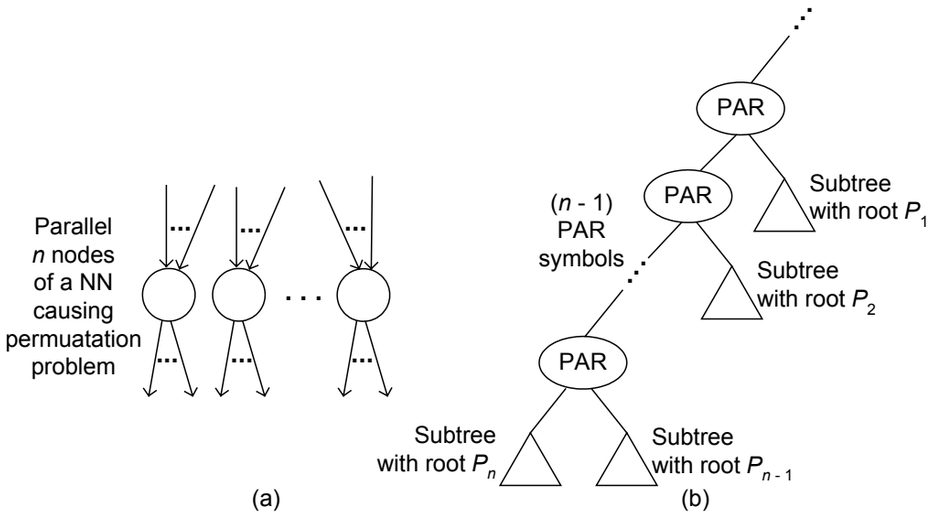

Fig. 4. Permutations of nodes in an ANN. (a) part of a ANN shows the nodes that may have any permutations. (b) CE of that part of ANN.

The n nodes in Figure 4(a) can be oriented in any permutation each give a ANN with the same behavior. Since these nodes can be arranged in $P(n, n) = n!$ ways, we have $n!$ ANNs all having same phenotype.

Now, the Random Selection Group (RSG) of the top PAR symbol in Figure 4(b) is made of the non-PAR symbols that are the roots of the subtrees shown in triangles. Subtree i is rooted with a program symbol P_i, *which* is not a PAR symbol. Hence, RSG constitute of P_1, P_2, P_3, . . . P_n (along with corresponding subtrees). According to Property 3, while executing the first right child we can choose any of the elements from RSG in n ways. For the next right child we have the remaining $n - 1$ elements from RSG leaving $n - 1$ ways to choose with. Thus continuing up to the last child we are left with 1 way for the last child. So, there is a total $n \times (n - 1) \times (n - 2) \times \ldots \times 1 = n!$ ways to select. It means that the CE of Figure 4(b) can represent all $n!$ ANNs that are of same phenotype. Thus, permutation problem is resolved by our modified CE scheme □

E. Other Properties of Modified Cellular Encoding

This subsection summarizes other properties of MCE.

a) When there is only one input link on a given cell, the CUT operation is not applicable for that cell [1, 3]. CUT cannot also be applied on a link when this is the only output link from the parent cell (or from input). Because if we allow it, the parent cell should be deleted after the operation developing a new PST that can be generated in another (simpler) way arising permutation problem.

b) MRG symbol cannot be applied on the first level of the ANN i.e. on the cells that are directly connected to the inputs. Like CUT, when there is only one input link on a given cell, the MRG operation is not applicable for that cell [1, 3].

c) The parameters of CUT and MRG can be used as modulus n+1 where n is the number of current input links of the cell now considering. If so, we need not worried or be informed about the number of current input links when we set the parameter.

d) First symbol (root) of PST must be PAR, SEQ or END.

3 THE GENETIC SEARCH SCHEME

In this section the effects of the genetic operator crossover upon the MCE encoded ANNs are discussed. The algorithm to realize the PST is also presented.

A. Crossover on Modified Cellular Encoding

This subsection illustrates the rules and effects of the genetic operator crossover on our modified CE.

While growing of a ANN from a PST, each cell transformed to a neuron when its reading head ends at an END symbol. This necessarily means that for each neuron there must have an END symbol, which is a leaf of the PST. In fact, the number of ENDs in a PST is equal to or greater than (some neurons of corresponding END leaves may be removed by MRG operation) the number of neurons in the ANN. To avoid the risk of changing the structure of ANN drastically by crossover effect, we allow crossover on last level (leaf) or second last level. That is, the roots of the two subtrees chosen form a pair of CE for crossover must have all children as END or no children. If the root of a subtree is a binary symbol (i.e. PAR or SEQ) it has two children both of END symbols. If the root is a unary symbol (i.e. CUT or MRG) it has single child END. If the root is END itself it has no children. But both of the subtrees chosen for exchange cannot be same since in that case the crossover would have no effect. The following table shows all possible combinations of crossover and their effects.

Effects of Crossover		***Replaced by***				
		PAR	*SEQ*	*CUT*	*MRG*	*END*
To be replaced	*PAR*	NC	OCN	CD, ND	OC, ND	ND
	SEQ	OCN	NC	CD, ND	OC, ND	ND
	CUT	CA, NA	CA, NA	OC or NC	OC, ND*	CA
	MRG	OC, NA	OC, NA	OC, NA*	NC or OCN	OC, NA*
	END	NA	NA	CD	OC, ND*	NC

Here,

NC ≡ no change in the architecture of corresponding neural networks
NA ≡ node addition
ND ≡ node deletion
CA ≡ connection addition
CD ≡ connection deletion
OC ≡ new orientation of connection
OCN ≡ new orientation of both connection and node
* effect may or may not takes place, depends on orientation of connections and the parameter of the corresponding symbol

It is clear from the above table that all sorts of modifications of ANNs (for example addition and / or deletion of node and / or connection) can be happened from crossover. Since we allow crossover only at the lowest level or immediate above it, massive changes in the architecture at a time cannot occur.

Now we put across the rules for applying the genetic operator crossover on our modified CE so that its properties are not destroyed even after crossover operation. First, we should remember that crossover cannot be applied on the roots of PSTs, it must be applied on two different subtrees. If these two subtrees are same crossover would have no effect. When applying crossover between two cellular encodings CE1 and CE2 by exchanging two random subtrees ST1 (from CE1) and ST2 (from CE2), the following three situations may arise.

i) None the parents of ST1 and ST2 is PAR.
ii) One of the parents of ST1 and ST2 is PAR.
iii) Both the parents of ST1 and ST2 are PAR.

In case (i), crossover is just applied accordingly.
In case (ii), let, without loss of generality, the parent of ST1 is PAR and the parent of ST2 is not PAR. The sub-cases can occur as follows:

a) Both the roots of ST1, ST2 are PAR
b) Both the roots of ST1, ST2 are not PAR
c) The root of ST1 is PAR, the root of ST2 is not PAR
d) The root of ST1 is not PAR, the root of ST2 is PAR

In sub-case (a) crossover is not applied. For other sub-cases it is applied accordingly. After crossover is done, the ordering described in Property 1 is established among the children of the parent of ST2 in CE1.
In case (iii), the sub-cases can occur as follows:

e) Both the roots of ST1, ST2 are PAR
f) Both the roots of ST1, ST2 are not PAR
g) The root of ST1 is PAR, the root of ST2 is not PAR
h) The root of ST1 is not PAR, the root of ST2 is PAR

In sub-case (e) crossover is not done. For other sub-cases, crossover is applied accordingly but the ordering described in Property 1 is established among the children of both the parent of ST2 in CE1 and the parent of ST1 in CE2.

B. Algorithm for Converting CE to Direct Encoding (Realization of PST)

Input: the PST to be executed.
Output: the matrix representation (direct encoding) of the ANN.

$m \equiv$ number of input nodes
$n \equiv$ number of output nodes
$N \equiv$ maximum number of hidden nodes allowed in the ANN
$M \equiv$ a matrix with $m + N$ rows and $N + n$ columns; $M[i, j] = 1$ means that there is a connection from i-th node to j-th node, $M[i, j] = 0$ means that there is no connection from i-th node to j-th node

Initialize a FIFO with the root of the PST as the only entry
Initialize all the entries of the matrix M to 0
while (head of the FIFO $\neq$ NULL)
case (head of the FIFO)
PAR: Copy the inputs and outputs of the current i-th entry (cell) in a new j-th entry (cell) of the matrix. Enter both of the entries (cells) i, j at the end of the FIFO in random order.

SEQ: Remove the outputs of the current *i*-th entry and copy these to a new *j*-th entry of the matrix; Set $M[i, j] = 1$. Enter entry *i* at the end of the FIFO, then enter entry *j* in the FIFO.

CUT *x*: Let the *x*-th input link to the current entry *i* is *j*. Set $M[j, i] = 0$. Place the entry *i* at the end of the FIFO.

MRG *x*: Let the *x*-th input link to the current entry *i* is *j*. If *j*-th entry has only one output link and its reading head not yet end, enter the current entry *i* at the end of the FIFO. Nothing changes in the matrix (this action is like WAIT). Otherwise, set $M[j, i] = 0$ and copy all the input links into entry *j* to the entry *i*. Also if *j*-th entry has no more output link, remove all the input links into entry *j*. Place entry *i* at the end of the FIFO.

END: Do nothing. Purge the reading head of the cell and convert it to a node of ANN.

end *case*

end *while*

The number of symbols *k* executed by the algorithm is the number of symbols present in the PST and $k = O(m + N + n)$. Let, PAR, SEQ and MRG handle on an average *l* number of connections (entries of the matrix) that depends on $O(m + N)$. So in the worst case, the cost for executing the above algorithm is $O(lk)$.

4 CONCLUSIONS

This paper presents a new indirect encoding method for ANNs in EP-based system. We modify [3] the CE scheme to avoid the permutation problem suffered by many evolutionary artificial neural network (EANN) systems. We choose a close and complete set of program symbols to generate the PSTs, i.e. the genotypes of ANNs. We exclude some symbols and change the functionality of other symbols. We also impose new restrictions on their appearances in the PSTs. These upgradations of CE direct us to a permutation problem free encoding.

Consequently, we can employ the genetic operator crossover on the genotypes of ANNs in the evolutionary system. Here, crossover tries all kinds of evolutions, i.e. deletion or addition of nodes or connections. To reduce the drastic change of architecture (and behavior) from parents to their children, we allow crossover at the lower levels of PSTs with higher probability. A hidden node is not added to an existing ANN at random, but through splitting an existing node by means of an additional program symbol to its PST. The proposed genetic search algorithm in this paper implements these strategies. It evolves ANN architectures and weights simultaneously, which is closer to the Lamarckian evolution [2]. Another significant improvement is the reduction of the number of user specified parameters.

Since our approach searches a much larger space than that searched by most other constructive or pruning algorithms and thus may require longer computation time, one of the future improvements would be to reduce evolution time. This can be achieved if the cellular encoded genotype can directly be incorporated with the training phase.

REFERENCES

[1] Frédéric Gruau, *Neural network synthesis using cellular encoding and the genetic algorithm*, Ph.D. Thesis, Ecole Normale Supérieure de Lyon, 1994.

[2] Xin Yao and Yong Liu, *A new evolutionary system for evolving artificial neural networks*, IEEE transactions on neural networks, vol 8, no 3, pp 694-713, 1997.

[3] Talib Hussain, *Cellular encoding: review and critique*, Queen's University, July 19, 1997.

[4] P. J. B. Hancock, *Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification*, Proc. of the Int'l

Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92) (D. whitley and J. D. Schaffer, eds.), pp. 108-122, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[5] Philipp Köhn, *Genetic encoding strategies for neural networks*, Master's thesis, University of Tennessee, Knoxville, IPMU, 1996.

[6] Frédéric Gruau, Darrell Whitley and Larry Pyeatt, *A comparison between cellular encoding and direct encoding for genetic neural networks*, Proceedings of the First Genetic Programming Conference, p. 81-89, 1996.

[7] Marko Grönroos, *A comparison of some methods for evolving neural networks*, Proceedings of GECCO'99, vol. 2, Morgan Kaufmann Publishers, San Francisco, California, 1999.

[8] S. E. Fahlman and C. Lebiere, *The cascade-correlation learning architecture*, Advances in Neural Information Processing Systems 2 (D. S. Touretzky, ed.), pp. 524-532, Morgan Kaufmann, San Mateo, CA, 1990.

[9] J. P. Nadal, *Study of growth algorithm for a feedforward network*, International Journal of Neural Systems, vol. 1, pp. 55-59, 1989.

[10] N. Burgess, *A constructive algorithm that converges for real-valued input patterns*, International Journal of Neural Systems, vol. 5, no. 1, pp. 59-66, 1994.

[11] R. Setiono and L. C. K. Hui, *Use of a quasi-newton method in a feedforward neural network construction algorithm*, IEEE Trans. on Neural Networks, vol. 6, no. 1, pp. 273-277, 1995.

[12] R. Reed, *Pruning algorithm – a survey*, IEEE Trans. on Neural Networks, vol. 4, no. 5, pp. 740-747, 1993.

[13] P. J. Angeline, G. M. Sauders and J. B. Pollack, *An evolutionary algorithm that constructs recurrent neural networks*, IEEE Trans. on Neural Networks, vol. 5, no. 1, pp. 54-65, 1994.

[14] David E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, published by Pearson Education (Singapore) Pte Ltd, ISBN 81-7808-130-X, Fifth Indian Reprint, pp. 10-14, pp. 236-238, 2002.

[15] Simon Haykin, *Neural networks: a comprehensive foundation*, published by Prentice Hall International, Inc., Upper Saddle River, New Jersey 07458, ISBN 0-13-908385-5, pp. 161-175, 1999.

[16] C. M. Friedrich and C. Moraga, *An evolutionary method to find good building blocks for architectures of artificial neural networks*, Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems, pp. 951-956, 1996.

[17] F. Gruau and D. Whitley, *Adding learning to the cellular development of neural networks: evolution and the baldwin effect*, Evolutionary Computation, vol. 1, pp. 213-233, 1993.

[18] D. B. Fogel, *Evolutionary computation: towards a new philosophy of machine intelligence*, IEEE Press, NY 10017-2394, 1995.

[19] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial intelligence through simulated evolutionary*, New York, NY: John Wiley & Sons, 1996.

[20] D. Whitley, T. Starkweather, and C. Bogart, *Genetic algorithms and neural networks: optimizing connections and connectivity*, Parallel Computing, 14: 347-361, 1990.