MBEANN: Mutation-Based Evolving Artificial Neural Networks

Yoshiyuki Matsumura[2], and Kanji Ueda[3]

[1] Hiroshima University, 1-4-1, Kagamiyama, Higashi-Hiroshima, 739-8527, Japan
[2] Shinshu University, 3-15-1, Tokida, Ueda, Nagano, 386-8567, Japan
[3] The University of Tokyo, 5-1-5, Kashiwanoha, Kashiwa, Chiba, 277-8568, Japan

Abstract. A novel approach to topology and weight evolving artificial neural networks (TWEANNs) is presented. Compared with previous TWEANNs, this method has two major characteristics. First, a set of genetic operations may be designed without recombination because it often generates an offspring whose fitness value is considerably worse than its parents. Instead, two topological mutations whose effect on fitness value is assumed to be nearly neutral are provided in the genetic operations set. Second, a new encoding technique is introduced to define a string as a set of substrings called operons. To examine our approach, computer simulations were conducted using the standard reinforcement learning problem known as the double pole balancing without velocity information. The results obtained were compared with NEAT results, which is recognised as one of the most powerful techniques in TWEANNs. It was found that our proposed approach yields competitive results, especially when the problem is difficult.

1 Introduction

Artificial evolution has been proven to be a promising approach to artificial neural networks (ANNs) in complex reinforcement learning tasks [5][3]. As discussed in [3][4], evolving artificial neural networks (EANNs) are faster and more efficient than reinforcement learning methods [6] for some representative benchmark problems. The reason is presumed to be that EANNs can search high-dimensional and continuous learning space more efficiently than other approaches. In addition, considering that ANNs can adapt to time series problems using the memory mechanism realised by recurrent synaptic connections, artificial evolution of ANNs would be a natural choice for learning non-Markovian tasks, a classification to which many interesting problems belong.

There has been a great deal of interest in EANNs. A good summary of EANNs up until 1999 can be found in [11]. Traditionally, EANNs are classified into the following three categories, according to their network structure:

- the network structure is fixed and the connection weights are evolving
- the network structure is evolving and the connection weights are trained by learning

F. Almeida e Costa et al. (Eds.): ECAL 2007, LNAI 4648, pp. 936–945, 2007.

– the network structure and the connection weights are evolving simultaneously

In the rest of this paper, we consider only the last case, which is called topology and weight evolving artificial neural networks (TWEANNs) [8]: of the three categories, artificial evolution plays the most important role in this one.

TWEANNs have several fundamental problems. One is that there are no generally effective guidelines for encoding a network structure into the form of a genotype. Currently, each researcher uses his own encoding method. A more serious problem is that the crossover operator cannot simply be applied to two individuals whose genetic information is different in length. Even when the crossover is applied to two individuals in a sort of brute force manner, the generated offspring often have much worse fitness values than the parents. As a result, most offspring would not survive into the next generation. Another factor that complicates the situation is that there is no effective theory for how to prepare the initial individuals. From the viewpoint of the crossover problem, it does not seem to be a good approach to provide the initial population with random topologies. Conversely, it seems to be inappropriate for the genetic search for all the individuals to have the same topology.

Many TWEANN approaches such as GNARL[1], EPnet[12] and ESP[3] have been proposed thus far. However, the authors regard the most impressive approach to be Neuro-Evolution of Augmenting Topologies (NEAT) [8], because NEAT solves the double pole balancing without velocity information (DPNV) problem, which is recognised as one of the difficult benchmark problems for reinforcement learning. The source code of NEAT is available at Stanley's homepage [10].

However, in our own experiments with NEAT, we encountered unwanted behaviour in the process of artificial evolution. We consider this to be unavoidable as long as crossover is adopted as the main genetic operator. Therefore, in this paper, we propose a novel method of TWEANNs called mutation-based evolving artificial neural networks (MBEANN), in which no crossover is used. This means that all genetic operations are applied to an individual independent of other individuals. Instead of crossover, two types of structural mutations are provided. Each is defined so as not to change the signal transfer, to make the effect on the fitness function value nearly or completely neutral.

The rest of this paper is organised as follows. Section 2 provides an overview of NEAT and explains how NEAT shows unwanted behaviour. Section 3 explains the details of our proposed MBEANN method. In Section 4, after introducing the general characteristics of DPNV, computer simulations are conducted to examine the effectiveness of MBEANN. The last section is the conclusion.

2 NEAT

NEAT is reported to be a very effective approach to the domain of TWEANNs [8] [9]. In this section, NEAT is briefly explained.

Encoding Method and Crossover. NEAT's genetic encoding scheme is designed to allow corresponding genes to be easily lined up when two individuals cross over during mating, since crossover is considered as the primary genetic operator. A genotype is represented by two lists, one for nodes and the other for synaptic connections. A node is represented by the following two components: one is the number that makes it uniquely identifiable for all the generations in a population; the other is the information that shows the type of layer that a node belongs to, i.e. the input layer, the hidden layer or the output layer. The genetic information for representing a synaptic connection is composed of two pointers to the nodes for both ends, the weight value, the flag (whether the connection is able to use it) and the innovation number. The innovation number is used for identifying the connection for all the generations in a population. Whenever a new synaptic connection is developed, this number is incremented and provided to the synaptic connection. Because of these innovation numbers, crossover can be applied to any two individuals without duplication or deletion of the genetic information from their parents, and the common part between them can be easily identified.

However, offspring generated by this crossover tend to have more genetic information than the parents. Therefore, the greater the difference between the parents, the larger the offspring that tend to be generated. This is a serious character because individuals can grow very large within several generations.

Speciation. In TWEANNs including NEAT, it is commonly observed that a structural change brings a harmful result, and therefore, a new offspring has a much worse fitness value than the parents. In NEAT, as a strategy to keep them alive for the next generation, a speciation technique with explicit fitness sharing [2] is adopted to protect new offspring from natural selection. Speciation also has the effect that crossover can be applied to only two individuals in the same species, i.e. to two individuals whose structural differences are within a compatible distance. The details are shown in [8].

This seems a reasonable and effective procedure at first glance. However, according to our computer simulations, the number of species almost always increased to the maximum number in earlier generations than we expected. Newly generated individuals tend to be protected from natural selection as a result of speciation. According to our NEAT computer simulations, we frequently encountered the unwanted situation in which there were too many species that include only one individual.

Initial Population. Typically, TWEANNs start with an initial population of random topologies to introduce a sufficient topological diversity. In contrast, NEAT biases the search towards minimal-dimensional spaces by starting out with a uniform population of networks with zero hidden nodes.

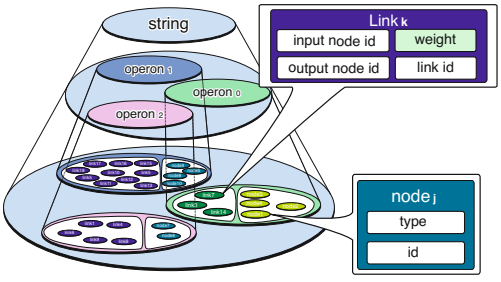


Fig. 1. The concept of genotype for MBEANN

3 Proposed Method: MBEANN

Although NEAT is a successful approach for TWEANNs, we still found the unwanted evolutionary behaviour that might originate from crossover as explained above. On the other hand, it might not be easy to propose a more effective strategy to reduce such side effects of crossover. Therefore, we propose a novel approach to TWEANNs in which no crossover is used. Our approach is called MBEANN, which stands for mutation-based evolving artificial neural networks. MBEANN adopts two types of structural mutations that may decrease little or have no fitness value. Therefore, speciation is not used in MBEANN.

3.1 Encoding Method

In the research of TWEANNs thus far, it does not seem that the topic of how to represent a network structure has been explored systematically. Each researcher has adopted his own representation method and special genetic operations[11][7], and no de facto standard is applied. Following this trend, we propose our original genotype representation as the answer to the question of how to design a robust genotype for a small change of genetic structure.

We considered the problem as follows. Assuming that an individual is a set of sub-networks each of which is independent of the others, we can expect that a small genetic change occurring in a sub-network would influence only a limited region. This might bring robustness against a genetic change. From this assumption, an individual is designed as a set of sub-networks, i.e. as a set of modules called operons. Fig. 1 shows the concept of the genotype that we designed. As shown in the figure, node information consists of the node type and the node identification number. Link information consists of the input node, the output node, the weight value and the link identification number. The two identification numbers should be unique to each individual. Note that they are not the same as the innovation numbers in NEAT, because they are unique only to an individual.

Thus, supposing that I is the maximum number of operons, a genotype *string* is formulated as follows:

$$string = \{operon_0, operon_1, , ..., operon_I\} \tag{1}$$

$$operon_i = \{\{node_j \mid j \in O_{Ni}\},\ \{link_k \mid k \in O_{Li}\}\} \tag{2}$$

where O_{Ni} is the set of node identification numbers in $operon_i$ and 0_{Li} is the set of link identification numbers in $operon_i$. Assuming that $operon_0$ holds only the input nodes, the output nodes and all the connections between them, string is composed of sub-networks $operon_i$, where $operon_i$ includes some nodes $node_j$ and the connections $link_k$ connecting two nodes in itself or a node in the $operon_i$ and a node of $operon_0$. As for an initial population, since MBEANN starts with the population consisting of the initial individuals having only one operon, $operon_0$, i.e. the minimal structure in which there is no hidden node.

3.2 Genetic Operators for Structural Evolution

Similar to NEAT, MBEANN starts with the minimal individuals. Hidden nodes and synaptic connections are obtained by two structural mutations with generations. The two genetic operators are defined as follows. They are designed to be nearly neutral or completely neutral with respect to the fitness value.

Add-Node Mutation. The add-node mutation is applied to each operon at a constant probability P_{add}. This mutation operates to remove one of the synaptic connections, which is randomly selected, and then adds a new hidden node and two associated synaptic connections. If one of the ends of the removed connection is connected to a node in $operon_0$, a new operon is provided to the newly developed sub-network. Fig. 2 illustrates the effect of the add-node mutation. When a synaptic connection W_1 is selected, the link is removed and a new node and two synaptic connections W_3 and W_4 are added to construct a new operon called $operon_i$.

In order not to change the signal transfer by this mutation, a_1 and a_2 in Fig. 2 should be equal. Therefore, supposing that S is the sigmoid function and the W_i is the weight value, is we consider the following condition:

$$S[w_1x_1 + w_2x_2] = S[w_3S[w_4x_1] + w_2x_2] \tag{3}$$

Thus, we get $w_1x_1 - w_3S[w_4x_1] = 0$. In addition, assuming the condition that $w_1 = w_3$ and $S[x] = 1/(1 + e^{\beta(\alpha-x)})$, we find that the following value $f(x_1)$ should always be small against α, β and w_4:

$$f(x_1) = x_1 - \frac{1}{1 + e^{\beta(\alpha - w_4x_1)}} \tag{4}$$

For the sake of simplicity, we define $w_4 = 1$. As for α and β, we set them as 0.5 and 4.0, respectively, considering the simplicity and the results of the preliminary experiments.

Add-Connection Mutation. The add-connection mutation is applied to each operon at a constant probability. Fig. 3 illustrates how this mutation works. When an operon is selected, a node in the operon is randomly chosen to make a random synaptic connection with a node in the operon or with a node in $operon_0$. The weight of the new connection is set to 0 so that it does not change the signal

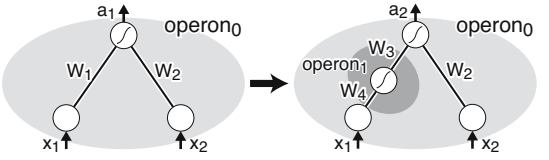


Fig. 2. Add-node mutation

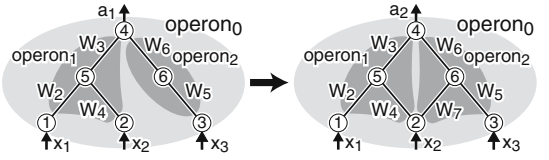


Fig. 3. Add-connection mutation

transfer. With this add-connection mutation, a node in $operon_i$ where $i > 0$ can be connected to a node in the same operon or a node in $operon_0$. In other words, the connection to a node in operon($j = i, j > 0$) is prohibited. Therefore, since each sub-network $operon_i$,$i > 0$ grows independent of the others, we can expect the functional modularity in an individual.

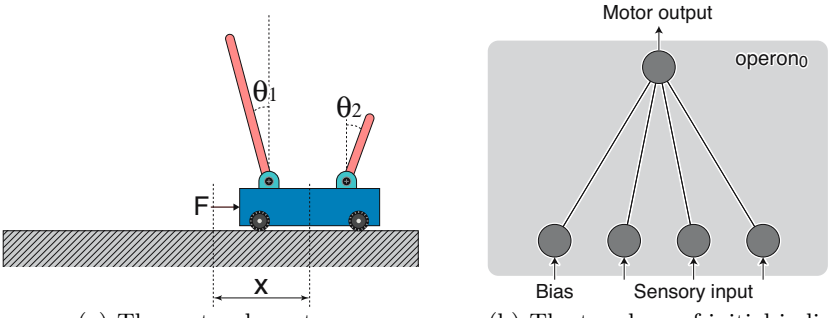
Synaptic Weight Mutation. As for weight mutations, we adopt the most popular mutation for real-coded genetic algorithms. That is, the weight mutation introduces a small change by adding a Gaussian random number of which the average is zero and the standard deviation is σ. In this paper, the value $\sigma = 0.05$ is adopted.

4 The Double Pole Balancing Problem

4.1 DPNV and Simulation Conditions

The performance of MBEANN is discussed with the benchmark problem called the double pole balancing without velocity information (DPNV) problem[3]. Two poles are connected to a moving cart by a hinge and the neural network must apply force to the cart to keep the poles balanced for as long as possible without going beyond the boundaries of the track. The system state is defined by the cart position x and the velocity $\dot{x}$, the first pole's position θ_1 and its angular velocity $\dot{\theta_1}$, and the second pole's position θ_2 and its angular velocity $\dot{\theta_2}$. However, $\dot{x}$, $\dot{\theta_1}$ and $\dot{\theta_2}$ are not used as inputs because the problem becomes more difficult; therefore the controller must utilise the time series information.

It is known that control is possible when the poles have different lengths to respond differently to control inputs. The computer simulations were conducted with four different ratios of the long pole length versus the short pole length, i.e.



(a) The cart pole system

(b) The topology of initial individuals

Fig. 4. Setting of the double pole balancing problem without velocity information

Table 1. Parameters for MBEANN and MBEANN-WO

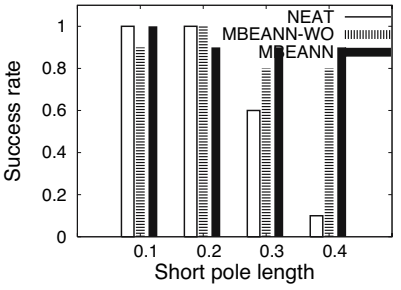
population size	1,000	weight mutation rate	1.0
tournament size	20	weight value in add-node	1
add-node rate	0.007	α in sigmoid function	0.5
add-connection rate	0.3	β in sigmoid function	4.9
final generation	1,000		

1:0.1, 1:0.2, 1:0.3 and 1:0.4. It is known that the control is more difficult when the difference between the pole lengths is smaller. All other details for problem settings for computer simulations were the same as in [8]. As for the comparison, the original NEAT source from Stanleys website[10] was used.

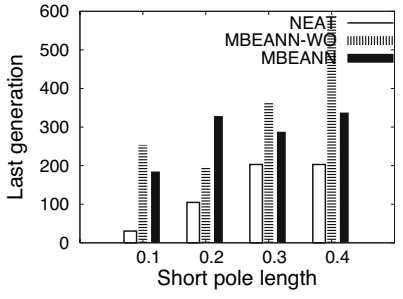
In addition, since the operon structure was employed for MBEANN based on our presumption that the modularity in genotype would work better, MBEANN without the operon structure (MBEANN-WO) was also conducted to compare the results with those of MBEANN to confirm our decision. MBEANN-WO can be easily realised by skipping the procedure of generating a new operon when add-node mutations are applied. The parameters for MBEANN and MBEANN-WO used in our computer simulations are summarised in Table 1.

4.2 Results

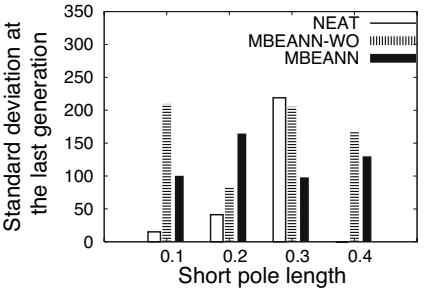
Fig. 5 summarises the results of ten independent runs of NEAT, MBEANN-WO and MBEANN. From Fig. 5(a), which shows the success rates, we found that all three approaches solve the problem with very high probabilities at easy conditions such as 1:0.1 or 1:0.2. However, it is clearly observed that NEAT's performance was getting worse as the problem difficulty increased. In particular, NEAT solved only once in ten runs under the most difficult condition of 1:0.4. At the same time, it was also found that MBEANN and MBEANN-WO performed much better than NEAT under difficult conditions.



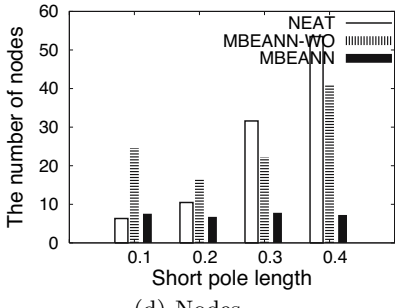
(a) Success rate



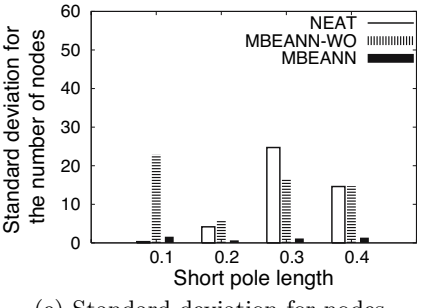
(b) Last generation



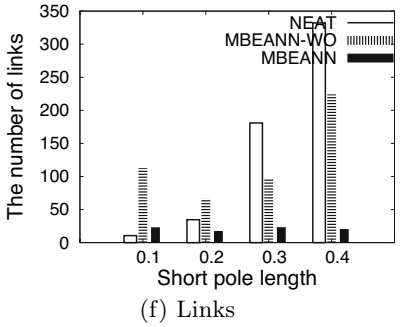
(c) Standard deviation at the last generation



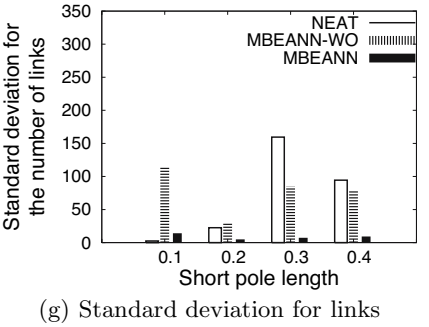
(d) Nodes



(e) Standard deviation for nodes



(f) Links



(g) Standard deviation for links

Fig. 5. Experimental results averaged over ten trials for NEAT, MBEANN-WO and MBEANN

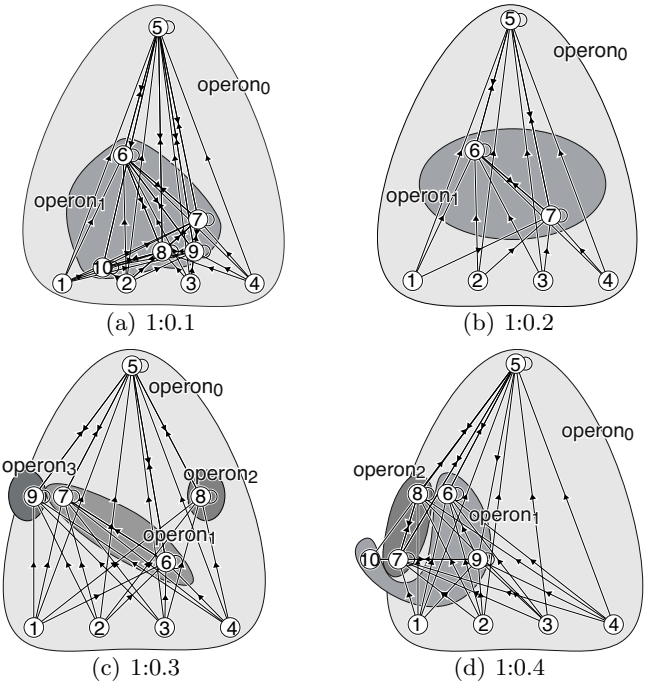


Fig. 6. Typical neural controllers at the last generation for MBEANN

For the rest of the graphs, the average and the standard deviation were calculated considering only the runs in which successful controllers were found before the final generation of 1,000. In addition, we call the generation in which a successful controller was found first in a run as the last generation, because it corresponds to the termination condition. Fig. 5(b) and (c) shows the average and standard deviations of the last generations. NEAT solves the problem of 1:0.1 within 30 generations. This result is much better than MBEANN or MBEANN-WO. However, when the pole length ratio is 1:0.3 or 1:0.4, NEAT requires about 200 generations to obtain a successful controller or cannot find a solution until the final generation. This implies that, as for NEAT, it might be better to stop the run and start a new run if a successful controller cannot be found before 200 generations. In the case of MBEANN-WO, the more difficult the problem, the more generations are required. MBEANN shows the most stable results of the three approaches. It seems that 400 generations are sufficient for MBEANN to solve DPNV for all four conditions.

Fig. 5(d) and (e) shows the average number of nodes in the controller at the last generation and the standard deviation of ten runs. Similarly, Fig. 5(f) and (g) shows the average number of links and the standard deviation. NEAT found a successful controller with only 6 nodes and 10 links at the condition of 1:0.1 in average. However, the number of nodes and links rapidly increases

with the problem difficulty. Finally, at the condition of 1:0.4, the controller had grown to reach the size of 53 nodes and 332 links. In the case of MBEANN-WO, the controller becomes gradually larger with the difficulty, but the speed seems moderate compared to the one of NEAT. On the other hand, the MBEANN's controllers are always similar in size for all four conditions. Controllers typically obtained are shown in Fig. 6. MBEANN needs about seven nodes and twenty links on average. Therefore, we can say that the importance of modularity in genotype has been inductively proven by the results of the computer simulations.

5 Conclusions

MBEANN was proposed as yet another approach to TWEANNs. Using the pole balancing without velocity information problem, its robustness as well as effectiveness were demonstrated. We believe that more investigation is required to improve the performance of TWEANNs, especially for the paradigm of evolutionary robotics[7], where TWEANNs are necessary to realise artificial evolution.

References

1. Angeline, P.J., Sauders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Networks 5(1), 54–65 (1994)
2. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, New York (1989)
3. Gomez, F., Miikkulainen, R.: Solving non-Markovian control tasks with neuroevolution. In: Dean, T. (ed.) Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, pp. 1356–1361. Morgan Kaufmann, San Francisco (1999)
4. Gomez, F., Miikkulainen, R.: Learning robust nonlinear control with neuroevolution. Technical Report AI02-292, Department of Computer Science, University of Texas at Austin, Austin, Texas (2002)
5. Grauau, F., Whitely, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding for genetic neural networks. In: Koza, J.R., et al. (eds.) Genetic Programming 1996: Proceedings of the First Annual Conference, pp. 81–89 (1996)
6. Kaebling, L.P., Littman, M., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence 4, 237–285 (1996)
7. Nolfi, S., Floreano, D.: Evolutionary Robotics. MIT Press, Cambridge (2000)
8. Stanley, K.O., Miikkulainen, R.: Evolving Neural Networks Through Augmenting Topologies. Evolutionary Computation 10(2), 99–127 (2002)
9. Stanley, K.O., Miikkulainen, R.: Competitive Coevolution Through Evolutionary Complexification. Journal of Artificial Intelligence Research 21, 63–100 (2004)
10. Stanley, K.O.: `http://www.cs.ucf.edu/~kstanley/`
11. Yao, X.: Evolving Artificial Neural Networks. Proceedings of the IEEE 87(9), 1423–1447 (1999)
12. Yao, X., Liu, Y.: A New Evolutionary System for Evolving Artificial Neural Networks. IEEE Transactions on Neural Networks 8(3), 694–713 (1997)