BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 59, No. 1, 2011
DOI: 10.2478/v10175-011-0001-6

Application of genetically evolved neural networks to dynamic terrain generation

Ł. CHOMĄTEK* and M. RUDNICKI

Institute of Information Technology, Technical University of Lodz, 225 Wólczańska St., 90-924 Łódź, Poland

Abstract. Real time terrain generation is a vital part in the development of realistic computer simulations and games. Dynamic terrain generation influences the realism of simulation, because its participants have to adapt to the current environment conditions. Dynamically generated primary terrain is transformed in order to reflect natural phenomena, such as thermal and water erosion, avalanches or glaciers. In this article a possibility of primary terrain transformation with application of artificial neural networks is shown. The networks are trained by evolutionary algorithms to solve a problem of a water erosion phenomenon. Obtained results show that application of such neural networks to this problem can significantly reduce the processing time needed to perform the process of modeling the natural phenomena.

Key words: evolving neural networks, water erosion.

1. Introduction

Real time terrain generation is a vital part in the development of realistic computer simulations and games. Dynamic terrain generation influences the realism of simulation, because its participants have to adapt to the current environment conditions; it is impossible when predefined terrains are used for the simulation. What is more, simulation of the natural phenomena can be applied to prevent its consequences.

In practice, the terrain generation process is performed in two phases. At first, the primary terrain is generated, usually by use of some fractal transformations. Dynamically generated primary terrain is transformed in order to reflect natural phenomena, such as thermal and water erosion, avalanches or glaciers. There is also the possibility to apply the models of disasters given by some mathematical models [1]. Other factors applied in such algorithms are utilized to construct the terrain's flora.

Solutions of the problem are usually based on the cellular automata [2, 3], where the cells represent the state of some part of the developed landscape. In [2] authors present the realistic model of water erosion. The state of each cell contains information about the terrain altitude, depth of the water and the amount of the eroded and deposited material. What is more some other factors like soil receptivity and sediment transport fluxes are held in the cells' state to improve the simulation realism. Results obtained in [2] show that such a model can be successfully applied to predict the results of water erosion in real world.

The water erosion implementation presented in this article was based on works [3, 4]. In the article the base terrain was described by means of the cellular automaton. Every cell stores the information not only about the altitude of the terrain in a given point but also about the amount of water and the sediment in the same point. The algorithm is an iterative process and each iteration consists of four phases:

- addition of certain amount of water to each cell,
- detachment of the part of the terrain, which is later floating on water,
- transportation of the sediment,
- evaporation and sedimentation.

For the purpose of this article the water erosion algorithm was implemented in the Matlab environment. The obtained results were used as a base for further research.

2. Terrain generation by means of the neural network

The idea of applying the neural network to process the data describing a terrain was employed in [5, 6]. In this case the aim of the neural networks application was not to optimize the speed of the algorithm operation, but to find the relations for certain practical issues. The results obtained by the cellular automata implementation of the water erosion algorithm, were utilized to prepare a neural network, which can perform water erosion simulation. Such a network was implemented in the Matlab environment. The application of this network has made it possible to conduct any iteration through a single computation of the new parameters' values for each cell of the prepared map.

2.1. The training and testing data. As it was mentioned above, each cell of the terrain is described by three parameters:

- the altitude of the terrain in the cell,
- the amount of water in the cell,
- the amount of the sediment.

To calculate the new values of those parameters, one need to know the values for the given cell and the four adjacent cells. Therefore, to perform the erosion, 15 parameters are needed. Updated values of the terrain height, the amount

*e-mail: lukaszch@ics.p.lodz.pl

of water and sediment for each operating cell are the result of the computations concerning erosion for each cell. Artificial neural network presented has 15 inputs and three outputs.

The training data was prepared through the notation of the parameters' values of the terrain for the randomly selected cells and their neighboring cells. For the primary terrain 500 patterns were prepared, then ten iterations of procedural erosion algorithm were arranged, during which appropriate amount of expected output data and 300 new input data from randomly chosen cells were added. As the initial amount of water in all cells did not equal zero and the network should learn this state as well, one decided to include data from different iterations.

The network was tested on a newly generated terrain, conducting several dozen iterations. The input in each iteration was the terrain generated by the neural network during preceding iteration.

2.2. The architecture and learning of the neural network. The neutral network chosen for this research was the multilayer perceptron [7]. Because of the fact that generally the unlimited values could appear as the outputs, the output layer is a linear layer. The neurons in other layers have sigmoid activation function. The calibration of the input data is realized through appropriate weights choice.

2.3. Obtained results. Exemplary results of the erosion computations are presented on the illustrations (Fig. 1). Like in the case of the procedural approach, it was noticed that a big amount of material was put aside at the foot of the hills located on the map. Unfortunately, sometimes various artifacts appeared on a map (for example sediment on the top of the mountain).

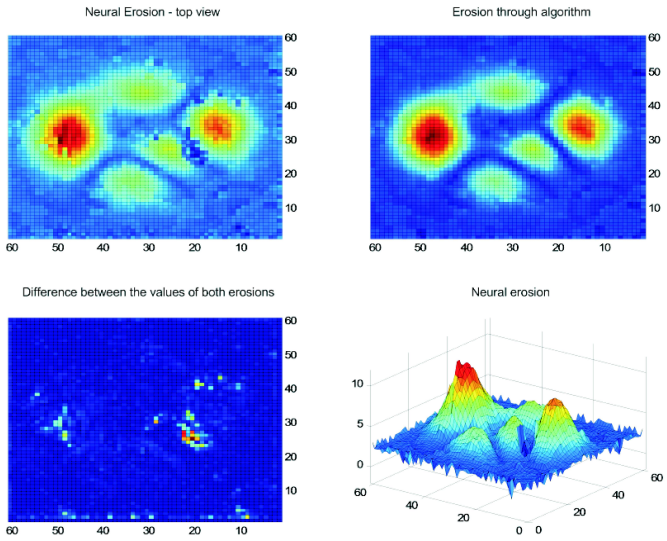

Fig. 1. Neural and procedural erosion

Remarkable shortening of the algorithm operation time is worth mentioning when talking about the attempt to engage artificial neural network against the prepared procedural implementation. Exemplary results are shown in the Table 1.

Table 1
Results for different SGA parameters (direct encoding)

SGA Parameters		Results [Threshold 0.05]		Results [Threshold 0.10]	
Mutation probability	Crossover probability	Number of weights	Iteration	Number of weights	Iteration
0.05	0.6	176	1290	76	1396
0.1	0.6	180	2890	74	1825
0.05	0.7	181	2970	70	857
0.1	0.7	175	1030	72	657
0.05	0.8	170	1540	71	1853
0.05	0.9	177	1540	83	1906
0.05	0.5	178	1220	75	1510
0.05	0.4	181	1725	77	1441
0.05	0.1	182	2400	71	1608

3. The optimization of the network architecture with the use of a genetic algorithm.

Empirical choice of the optimal amount of neurons in the hidden layers of the neural network is a time-consuming process. There are many ways for the automatic optimization of the architecture of the neural network by using the genetic algorithms.

3.1. Earlier attempts. One of the simplest ways is binary encoding of the information about the links between the neurons [8] Such network can be trained with any known method and this causes a remarkable computing addition. Another attempt to encode the architecture of the network in the chromosome was presented in the work referred to in [9]. A binary encoding with variable number of weighed bits was used which allowed the shortening of the chromosome at the small weigh value. The authors of [10] [11] proposed an innovative method which would create networks that were not multilayer perceptrons as generally acknowledged. The information about the connections between a given neuron and other neurons was encoded in the chromosome. Storing the weights in the chromosome was also possible. Networks prepared in such a way were taught by a small number of epochs by means of the reverse backpropagation algorithm in order to check if they have a chance to learn how to solve the problem. In [12], the network was encoded as the set of paths where the input neuron is the first element of each path and the output neuron is its' last element. Between the beginning and the end of a given path any hidden neurons can be found (cyclical connections are exceptionally permitted). Suggested mutation operators were related to the change of the existing paths. Crossover was based on the change of the fragments of the existing paths. In [13] it was suggested to encode the neural network in the form of a tree. In the beginning of the algorithm the network has one hidden neuron (the structure of this network develops during the algorithm operation). Subsequently new neurons or layers are added. The author suggested

many additional operators which are able to enlarge the generated model of network, for example, adding the recurrent connections.

3.2. Suggested solution. Genetic algorithms used to prepare the neural network for the purpose of this work, were based on encoding the weights of the neurons in the chromosomes. Therefore, the algorithm operation enabled both to find the desired architecture of the network and its automatic learning.

In order to solve the problem the HFC genetic algorithm was used. The parameters for such algorithm are: the number of the islands, the number of the individuals on each island, migration frequency, the number of the migrating individuals. In the stage of migration the relocation of the individuals to another islands comes as follows:

- on the island I (potentially least developed) the best-fit individuals are relocated to island II, the least-fit individuals are substituted with the new ones with the random gene values,
- on the intermediate islands the best-fit individuals are relocated to higher islands, the least-fit ones are substituted with the individuals from the lower island,
- on the last island the least-fit individuals are substituted with the ones from the penultimate island.

It was decided to encode the weights of the evolving neural network in the chromosome. The attempt mentioned earlier allowed to omit the learning process of the network through a different method as well as enabled to change its architecture dynamically. The computation of the fitness function values for each individual was based on checking the output errors for randomly chosen data from the training set. The individuals representing the network amounting smaller error had higher value of the fitness function.

Two types of genes were checked in this issue:

- direct weights encoding – the genes are the floating-point numbers,
- direct encoding with the fitness index – the gene is a pair of floating-point numbers consisting of the weight value and the fitness index.

During the HFC operation, iterations of a simple genetic algorithm with the mutation and crossover operators appropriate for the encoding were realized on each island. The methods of encoding as well as the genetic operators are described in detail in further subsections.

At some point in the initiation of the chromosome the networks are created, where each neuron from every layer apart from the output layer is connected to every neuron from the deeper layer. The chromosomes are initiated by the random values according to the acknowledged encoding.

3.3. Direct encoding. The encoding of the network weights as the floating-point numbers is more natural than binary encoding because of the possibility of a direct mapping the gene to the weight value. A simple genetic algorithm using this encoding utilizes the single or multiplepoint crossover, whereas the mutation takes place through adding a random number to the gene value. The mutation happens with a certain probability for each gene, like in a simple genetic algorithm. The tests were performed for two mutation methods: mutation using uniform distribution (in the range $-1 \ldots 1$) and normal distribution (expected value 0, standard deviation 1).

The genes are initiated with the values from $-1 \ldots 1$ range.

3.4. Direct encoding with the fitness function. In this encoding method the fitness index which changes the mutation size in relation to the maximum size from the previous encoding was utilized. Each gene stores the information about the current weight value (real number) as well as about the fitness function value (a number form the $0.05 \ldots 5$ range). The operation of the mutation operator in this method is based on conducting the mutation for the randomly chosen genes with certain probability, according to the formula:

$$g_i^{t+1} = g_i^t + \text{rand} c_{g_t}, \tag{1}$$

where g_i^{t+1} – the value of the subsequent chromosome gene in the iteration $t+1$, g_i^t – the current value of the subsequent chromosome gene, rand – a random number generated through the uniform distribution from the range of $-1 \ldots 1$, or through the normal distribution, c_{g_t} – fitness index of the subsequent gene in the current iteration.

After the mutation is conducted, new fitness function value for the chromosome is computed. If it is better than before, the mutation range for the mutated genes is increased. Otherwise, the range of those genes is reduced. In order to prevent too big an increase or reduction of the mutation indexes, they were limited to the values from the range $0.05 \ldots 5$.

The crossover operator was created on the basis of standard single or multiplepoint crossover, however, in the case of this encoding, if the genes are exchanged between the chromosomes, their fitness index values are set as 1. If the genes are not exchanged, their fitness index values remain the same.

The genes are initiated with the values from $-1 \ldots 1$ range, fitness index of each gene is set as 1.

4. Obtained results

In the table below the results gathered during the genetic algorithm action with binary encoding for fixed parameters HFC and floating parameters SGA are presented. In SGA a three-point crossover, tournament selection and uniform distribution at mutation were used. The research was conducted to test the optimal parameters of a simple genetic algorithm. Obtained results are gathered in Table 1.

From the analysis of the gathered data it is recognized that the application of the small probability of mutation is the most useful. Optimal value for the crossover probability comprises in the range of $0.5 \ldots 0.8$. During the genetic algorithm operation initially temporary decrease of the weights in comparison with the amounts of weights from the table was observed. It is probably caused by the application of small weights during initiation, which can be easily reset.

Table 2
Results for different HFC Parameters (direct encoding)

HFC Parameters				Results [Threshold 0.05]		Results [Threshold 0.10]	
Number of Islands	Population	Migration rate	Migration frequency	Number of weights	Iteration	Number of weights	Iteration
2	100	5	10	172	600	81	654
2	150	5	10	173	620	83	815
2	100	10	10	169	760	93	633
2	100	15	10	175	870	91	568
2	100	15	8	172	840	89	669
2	100	15	6	175	820	72	650
3	100	15	2	174	1120	110	335
4	100	15	2	173	1240	85	519

After finding the optimal values for a simple genetic algorithm which works on every island, the parameters concerning the amount of islands and migrations were tested. A simple genetic algorithm with the crossover probability of 0.5 and mutation probability of 0.05 was used during experiments. The mutation was realized with uniform distribution, the single-point crossover and the tournament selection were used. Results are shown in Table 2.

It turned out that the optimal number of the individuals on an island for the given parameters of the algorithm is about 50–100. The increase of the number of individuals did not bring about results in the form of the decrease of iterations needed to proper network preparation. The number of the individuals cannot be too small either, because the number of iterations needed to achieve a good result grows rapidly, whereas the number of individuals falls down.

The examination of the optimal number of the migrating individuals and migration frequency was also tested. It turned out that the algorithm achieves the best result when the number of the migrating individuals is not higher than 10% of all the individuals on the island. It was not possible to find out the relation between the result and the migration frequency, however, the similar good results were achieved for very small (two iterations) and relatively high (ten iterations) frequency. The increase of the number of islands did not significantly affect how quickly a solution was found, nor its quality. The research proved that with bigger number of the islands, higher migration frequency should be assumed.

4.1. A floating-point encoding with fitness function – results. The conducted research was based on finding optimal parameters of the SGA algorithm for set parameters of the HFC algorithm. The following parameters of the HFC algorithm were chosen:

– number of islands: 2,
– number of individuals on an island: 50,
– migration frequency: 10,
– migration rate: 5.

Obtained results are gathered in Table 3.

Table 3
Results for different SGA parameters (encoding with fitness index)

SGA Parameters		Results [Threshold 0.05]		Results [Threshold 0.10]	
Mutation probability	Crossover probability	Number of weights	Iteration	Number of weights	Iteration
0.15	0.4	178	1960	75	1682
0.05	0.5	184	1950	117	1068
0.05	0.6	188	1170	83	2382
0.05	0.7	177	1810	104	1946
0.15	0.7	186	6300	74	3084
0.1	0.8	182	2070	79	1492
0.05	0.9	184	2160	95	1439
0.1	0.9	186	2510	77	1370

The results show that with low weight reset threshold for optimal mutation values (0.1) the method with the fitness of weight change index finds a solution faster than the method without fitness. The solution is characterized by better quality for higher crossover index values than it was in the case of the method without fitness.

For higher values of the weight reset index, the encoding method with fitness index gives results faster than for lower values of weight reset index. It is worth mentioning that for higher values of weight reset index a higher mutation probability (0.1 or 0.15) can be assumed, which will result in better quality of the solution.

Further research was based on finding optimal parameters of the HFC algorithm. The following parameters of the SGA algorithm were tested:

– crossover probability: 0.8,
– mutation probability: 0.05.

Obtained results are shown in Table 4.

Table 4
Results for different HFC parameters (encoding with fitness index)

HFC Parameters				Results [Threshold 0.05]		Results [Threshold 0.10]	
Number of Islands	Population size	Migration rate	Migration frequency	Number of weights	Iteration	Number of weights	Iteration
2	50	5	10	176	1250	100	753
2	100	5	10	179	650	115	850
2	150	5	10	180	300	125	638
2	200	5	10	180	820	119	317
2	150	10	10	179	550	117	720
2	150	15	10	184	790	119	784
2	150	5	2	177	310	121	574
2	150	10	2	188	1890	124	457
2	150	10	4	182	770	114	645
2	150	10	6	182	910	110	688
2	150	5	8	184	820	111	384
3	150	5	2	183	449	118	517
4	150	5	2	181	610	105	783
5	150	5	2	185	470	116	550

The quality and speed of the HFC algorithm operation for various number of the individuals on an island were tested. For optimal value (150 individuals) the migration rate, then migration frequency and optimal number of islands were examined. It turned out that in case of small a value of the weight reset index the migration rate does not have the instance of an algorithm with the fitness of weight change index as well as it does not have significant influence on how fast the solution is found. When the migration happens more often it is recommended to limit the number of the migrating individuals. The increase of the number of islands does not significantly affect how fast the solution is found nor its quality. With higher values of the reset index, the weight reduction is bigger. It was not possible to find out the regularities connected to the result quality.

Comparing the efficiency of the HFC algorithm in both methods of encoding it is evident that the speed of finding a solution is higher in the case of the method with fitness. When it comes to the quality of the solution, both methods do not differ much.

The results achieved for the mediate encoding are much worse than presented here. The roulettewheel selection method, as well as the one-point crossover operator cause the decrease of the speed of the algorithm convergence.

4.2. Results according to the weight reset parameter. The research concerning the weight reset index for direct encoding with fitness for parameters recognized as optimal on the basis of earlier research was conducted:

- two islands,150 individuals on an island,
- migration frequency: 2,
- migration rate: 5,
- crossover probability: 0.8,
- mutation probability: 0.05.

Analogical research was also conducted for the encoding without fitness (Table 5). The reduction was evident even if the reset index was 0.1, however for higher values of this index the weights disappeared too fast and the algorithm could not find a solution

Table 5
Results for different values of weight reset index

Reset index	Number of weights	Iteration
0.06	177	750
0.08	163	853
0.09	155	741
0.10	126	1029
0.16	111	1015
0.17	112	549
0.19	99	1775
0.22	97	1427
0.23	87	1376

5. Conclusions

- For a given problem, thanks to the weights reduction, the speed of the neural network operation improved even up to 64%.
- The fitness of the weight change index during mutation results in the decrease of the number of iterations needed to finish the HFC algorithm operation.
- It is recommended to limit the number of islands to minimum in the HFC algorithm, because the bigger amount of the islands significantly affects the time devoted to single iteration.
- The application of the multiple-point crossover and tournament selection instead of singlepoint crossover and roulette-wheel selection method is more profitable.
- The distribution of the random numbers for the mutation operator does not significantly influence the found solution.

– It is essential to set a proper value of the weight reset index because, depending on the selected encoding, the algorithm might have problems with finding the solution or it will not work in an optimal way.

REFERENCES

[1] M. Gad-El-Hak, "The art and science of large-scale disasters", *Bull. Pol. Ac.: Tech.* 57 (1), 3–34 (2009).

[2] D. D'Ambrosio, S. Gregorio, S. Gabriele, and R. Gaudio, "A cellular automata model for soil erosion by water", *Physics and Chemistry of the Earth* B 26, 33–39 (2001).

[3] J. Olsen, "Realtime procedural terrain generation – realtime synthesis of eroded fractal terrain for use in computer games", *Oddlabs*, available at: http://oddlabs.com/download/terrain_generation.pdf (accessed 2009).

[4] B. Beneš and R. Forsbach, "Visual simulation of hydraulic erosion", *J. WSCG* 1, 79–86 (1994).

[5] G. Niedbała and K. Klejna, "Analysis of forecasting possibilities of soil displacements during ploughing with the use of classic statistical methods and artificial neural networks", *Agricultural Engineering* 2, 90 (2007), (in Polish).

[6] T. Behrens, H. Forster, T. Scholten, U. Steinrucken, E. Spies, and M. Goldschmitt, "Digital soil mapping using artificial neural networks", *J. Plant Nutrition and Soil Science* 168, 21–33 (2005).

[7] P. Szczepaniak and J. Protasiewicz, "Price prediction of the electric energy – regression versus neural approach", *J. Appl. Computer Science* 2 (15), 7–17 (2007).

[8] L. Marti, "Genetically generated neural networks", *Proc. Int. Joint Conf. on Neural Networks* 1, CD-ROM (1992).

[9] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks", *IEEE Trans. Neural Networks* 5 (1), 39–53 (1994).

[10] D. White, "GANNET: a genetic algorithm for searching topology and weight spaces in neural network design", *Lecture Notes in Coputer Science* 686, 322–327 (1993).

[11] W. Schiffmann, M. Joost, and R. Werner, "Application of genetic algorithms to the construction of topologies for multilayer perceptrons", *Proc. Int. Conf. Artificial Neural Nets and Genetic Algorithms* 1, 675–682 (1993).

[12] C. Jacob and J. Rehder, "Evolution of neural net architectures by a hierarchical grammar-based genetic system", *Proc. Int. Conf. on Artificial Neural Networks and Genetic Algorithms* 1, CD-ROM (1993).

[13] F. Gruau, D. Whitley, and L. Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks", *Genetic Programming Proc. First Annual Conf.* 1, 81–89 (1996).