Permutation Free Encoding Technique for Evolving Neural Networks

Anupam Das, Md. Shohrab Hossain, Saeed Muhammad Abdullah, and Rashed Ul Islam

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
{anupamdas,mshohrabhossain}@cse.buet.ac.bd,
saeed.siam@gmail.com, rislam101010@yahoo.com
http://www.buet.ac.bd/cse

Abstract. This paper presents a new evolutionary system using genetic algorithm for evolving artificial neural networks (ANNs). The proposed algorithm is "Permutation free Encoding Technique for Evolving Neural Networks"(PETENN) that uses a novel encoding scheme for representing ANNs. Existing genetic algorithms (GAs) for evolving ANNs suffer from the permutation problem, resulting from the recombination operator. Evolutionary Programming (EP) does not use recombination operator entirely. But the proposed encoding scheme avoids permutation problem by applying a sorting technique. PETENN uses two types of recombination operators that ensure automatic addition or deletion of nodes or links during the crossover process. The evolutionary system has been implemented and applied to a number of benchmark problems in machine learning and neural networks. The experimental results show that the system can dynamically evolve ANN architectures, showing competitiveness and, in some cases, superiority in performance.

Keywords: Permutation problem, genetic algorithm (GA), evolutionary programming (EP), artificial neural network(ANN).

1 Introduction

Artificial Neural Networks (ANNs) provide a practical way of representing a function such as a classifier when the input data is complex or noisy. Typically the ANN representation is learned by creating fixed network architecture with random connection weights, and then introducing training data to a backpropagation (BP) algorithm that adjusts the connection weights until the desired accuracy is achieved. This fixed-architecture BP learning method works well for simple functions of simple data. However, if ANNs are to be used to represent complicated functions of very complex data, the network structure will need to be more complex. Since the architecture is constrained, there is no opportunity for the learning algorithm to find a better architecture for the problem. BP becomes impractical if there are too many nodes or too many layers. BP tends to converge to local maxima.

F. Sun et al. (Eds.): ISNN 2008, Part I, LNCS 5263, pp. 255–265, 2008.

An alternate approach to ANN learning that may not have these disadvantages is the evolutionary algorithm. Evolution is less likely to get stuck in local maxima because of its strong tendency toward exploration. Evolution of ANN can be seen as a two-fold challenge; finding the topology of the ANN with optimal search space, and adjusting the weight values with a view to reaching that global optimum. Devising such an algorithm for a given problem has been a very challenging task. Many consequent problems have arisen in constructing such ANNs, yet to be unresolved. Consequently, the challenge is open for the ANN researchers for decades.

2 PETENN

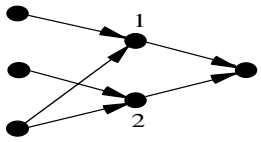
PETENN [1] uses a genetic algorithm for evolving ANNs. The encoding scheme is so designed to avoid permutation problem and make the whole process more efficient. The algorithm uses elitist recombination scheme. The recombination operator may be normal or mixed. Such operators ensure automation in the addition or deletion of nodes and/or links from the parents. As a result, the whole process is free from user interaction. Mutation operators are also applied randomly with a very small probability.

The steps of the algorithm are as follows:

1) Generate an initial population of N networks at random each of which is free from permutation problem.
2) Train each network partially for certain number of epochs using BP algorithm.
3) If *stopping criteria* is met, then stop the process and identify the best network. Otherwise go to step 4.
4) Perform a *random shuffle* on current population (networks) and pair them up. The recombination operator and mutation operator is applied for each pair to obtain two offspring. Ignore the offspring that introduces permutation problem.
5) A competition is performed between the offspring and their parents. The two winners are transferred to the next population.
6) Go to step 2.

2.1 Encoding Scheme

The main objective of the proposed encoding scheme is to ensure that neural networks with behaviorally equivalent structure will produce similar genotypic representations. This is a kind of direct encoding scheme where architecture is encoded. This paper concentrates on only a single hidden layer. So there are three layers of nodes: input, hidden and output. Each hidden node of an ANN has a bit pattern corresponding to the connections with the input and output nodes. One bit is needed for each of the input and output nodes. The value '1' means a connection exists and '0' means no connection. This bit pattern will produce



Hidden node	Bit String	Decimal Value
1	1011	11
2	0111	7

Genotype : | 11 | 7 |

Fig. 1. An ANN and its genotype in the proposed encoding scheme

a decimal value for that hidden node. Fig. 1 shows the encoding technique for an ANN with two hidden nodes. There are 3 input nodes and 1 output node. So 4 bits will be needed to encode each hidden node's connection structure. The first hidden node is not connected to the second input node but has links with all others. So the bit pattern is 1011 and the corresponding decimal value is 11. Similarly, the second hidden node's bit pattern is 0111 and decimal value 7. Now these two values are sorted and the genotype for the ANN is "7 - 11". So the structural information of each hidden node is encoded in this scheme.

2.2 Fitness Evaluation and Selection Mechanism

The fitness of each individual is solely determined by the inverse of an error function E suggested by [3]

$$E = 100\frac{O_{max} - O_{min}}{Tn}\sum_{t=1}^{T}\sum_{i=1}^{n}\left(Y\left(i,t\right) - Z\left(i,t\right)\right)^2 \ . \tag{1}$$

Here, O_{max} = maximum value of output coefficients, O_{min} = minimum value of output coefficients, n = number of output nodes, T = number of patterns, $Y\left(i,t\right)$ = actual outputs of nodes, $Z\left(i,t\right)$ = desired outputs of nodes.

The error measure is less dependent on the size of the validation set and the number of output nodes.

2.3 Recombination

The first step of recombination is selection of parents. Here "Elitist recombination" is used where the population is randomly shuffled and parents are selected by taking pairs of individuals. This corresponds to a uniform selection where each individual has exactly same probability of being selected as a parent. The recombination and replacement strategy are described in step 4 and 5 of the algorithm. It is found that instead of global competition, local competition is held and thus discouraging the greedy approach.

2.4 Recombination Operator

Two types of recombination operator are applied: normal and mixed. The recombination is applied to the hidden nodes of the parents.

Normal Recombination Operator: In this scheme, the number of crossover point is random and limited to a highest value. Alternate parts are taken from the parents to build up the offspring. Fig. 2(a) shows how child-1 and child-2 are produced from parent-1 and parent-2 using normal recombination operator.

Mixed Recombination Operator: In this scheme, the genotypes(hidden nodes) of the two parents are mixed by concatenation. Then a random crossover point is taken and the two portions correspond to the two offspring. Fig. 2(b) shows how child-1 and child-2 are produced using mixed recombination operator.

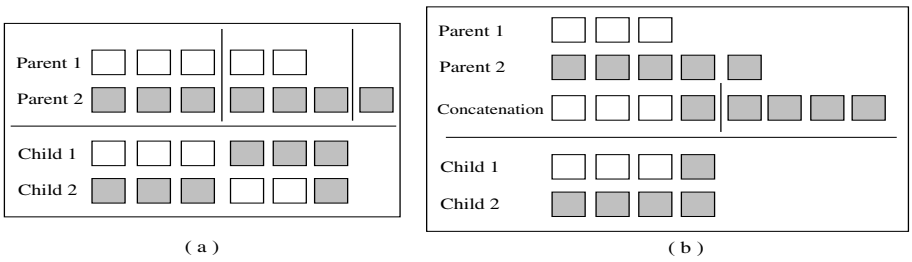


Fig. 2. Schematic representation of (a)normal recombination (b) mixed recombination

2.5 Automation

The recombination operators will ensure automation through connection (link) addition or deletion and node addition or deletion. The following sections describe the scenarios with examples.

Connection Addition / Deletion: In the proposed encoding scheme, the values correspond to how hidden nodes are related to their connections with the input and output nodes. So any change in the encoded value implies alterations in those connections. This scenario is applicable to the recombination operator where genotypes of the children are different from their parents. Such an example is shown in Fig. 3.

Node Addition / Deletion: Nodes will be added or deleted from the parent ANN if parents have different number of hidden nodes and multiple crossover points are used. This is shown in Fig. 3(b) where parent-1 has three hidden nodes and parent-2 has five hidden nodes. Here two crossover points are used. Now after recombination both children have four hidden nodes.

2.6 Mutation

After recombination, the mutation operator may be applied on the offspring before a competition takes place among the parents and their offspring. The

mutation operators are applied with certain probabilities. The following mutation operators can be applied on the offspring: Firstly, a new hidden node may be added to the network. The links and their corresponding weights of this node with the input and output nodes are randomly generated. Secondly, a hidden node of the network may be randomly deleted. The probability of adding a node is more than that of deleting it. Thirdly, the network may be mutated by deletion of a randomly chosen link. And finally, the weight of a randomly selected link may be changed within a certain range.

2.7 Solution to the Permutation Problem

In general, the evolution of ANN architecture suffers from the permutation problem [4], [5] which is caused by the many-to-one mapping from genotypes to phenotypes. It is obvious that if two hidden nodes have the same bit patterns and hence same values, they have similar connections with the input and output nodes. Now consider two structurally similar ANNs with their hidden nodes in different order (Fig. 4). The bit patterns for their hidden nodes will also be similar but in different order. In our encoding scheme, the values are sorted to get the final sequence. So those two ANNs will produce one-to-one mapping from genotypes to phenotypes, thus solving the permutation problem.

3 Experimental Studies

This section evaluates PETENN's performance on some of the well known benchmark problems. These problems have been the subject of many studies in NNs and machine learning.

3.1 The Medical Diagnosis Problems

PETENN was applied to four real-world problems in the medical domain e.g., breast cancer problem, diabetes problem, heart disease problem, and thyroid problem. All the datasets were obtained from the University of California, Irvine (UCI) machine learning benchmark repository, available at http://ftp.ics.uci.edu/pub/machine-learning-databases/. The characteristics of the data sets are summarized in Table 1.

Table 1. Characteristics of the Data sets

Data set	No. of examples	No. of input attributes	No. of output classes
Breast cancer	699	9	2
Diabetes	768	8	2
Heart disease	303	35	2
Thyroid	7200	21	3

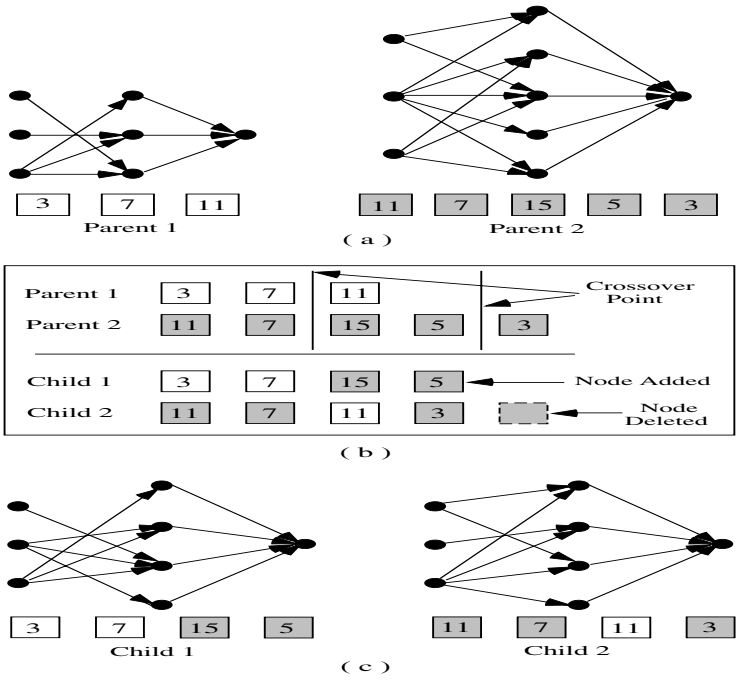


Fig. 3. (a) The parents used in recombination (b) Recombination using multiple crossover points (c) The two offspring produced

3.2 Experimental Setup

All datasets used have been partitioned into three sets: training set, validation set, and testing set (Table 2). In most case, the partitioning is 50%-25%- 25% except for the thyroid problem because it contains huge learning data. For each problem, the input attributes are normalized between 0 and 1 (real) and output attribute is binary valued. The output attributes of all the problems were encoded using a 1-of-m output representation for m classes.

The genetic population consists of random number of individuals. Each individual is an ANN of one hidden layer as it is sufficient to solve all non-linear problems [6],[7]. Initially, number of nodes in the hidden layer is uniformly chosen between minimum value =(input+output)∗0.5 and maximum value =(input+output)∗1.5. Initially, connections between the hidden layer and input layer are randomly created. The output layers are fully connected. Initial weights are assigned between -1.0 to 1.0. Activation function used is

$$sigmoid = \frac{1}{(1 + e^{-x})} \ . \tag{2}$$

Learning rate of the back-propagation algorithm is set to 0.5. The number of epochs is set to 10-20 for initial training and 5-10 for partial training. The

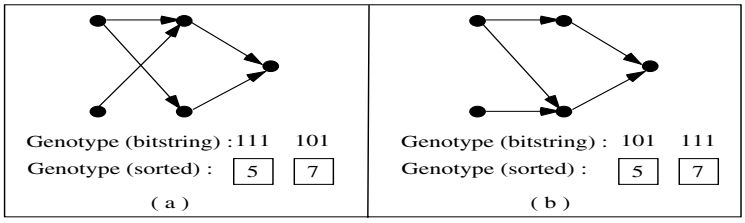


Fig. 4. Similar ANNs with different orientation

Table 2. Dataset partitions

Data set	Training data	Validation data	Test data
Breast cancer	350	175	174
Diabetes	384	192	192
Heart disease	152	75	76
Thyroid	2514	1258	3428

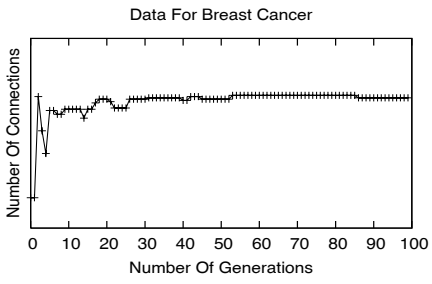
stopping criterion for the algorithm is based on a predefined *threshold value*. This value is different for different problems. If the average error does not fall below the threshold after maximum number of iterations (300 or 500), the evolution is stopped at that point.

3.3 Experimental Results

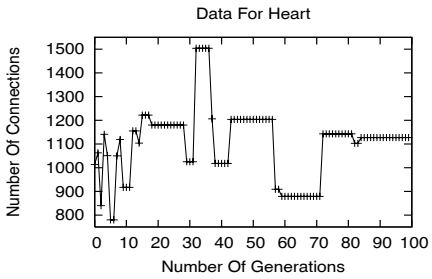
This section presents the results of PETENN over 30 independent runs on the four different problems. The error in the table refers to the error defined by the fitness function E (1). The error rate refers to the percentage of wrong classifications produced by the evolved ANN's.

Table 3 summarizes the average, standard deviations(SD), minimum, maximum number of hidden nodes, connections and generations for each set of problems. It also shows the mean, standard deviation, minimum and maximum value of validation and testing errors for the four problems. Most of the errors are quite small.

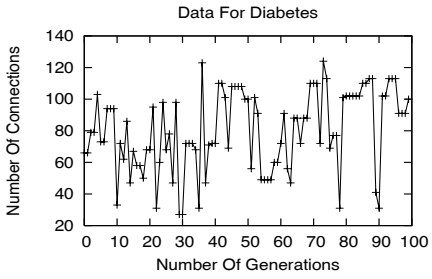
In order to observe the evolutionary process in PETENN, Figs. 5–6 show the evolution of the numbers of connections and the error (inverse of accuracy) of the best ANN over 30 runs for the four medical diagnosis problems. The evolutionary processes are quite interesting. The number of connections in the ANN randomly decreases and increases through out the process signifying the recombination operator. The recombination operator actually serves as the process of exploring the population to find the best suitable network. Similarly as the generation proceeds, the amount of error decreases significantly which implies that the algorithm is moving towards the best structure. As we can see in the figures that error drops quite quickly to the lowest value showing the effectiveness of the recombination and mutation process.



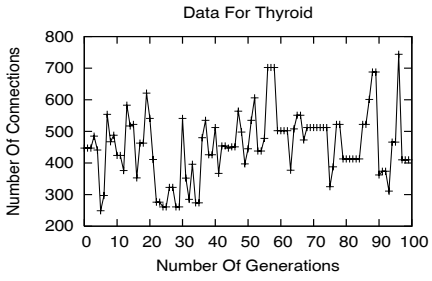
(a)



(b)

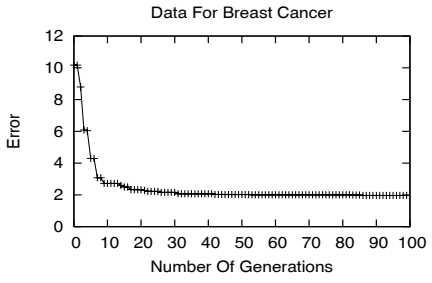


(c)

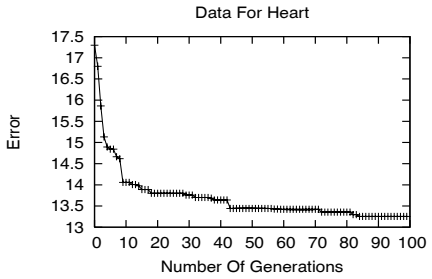


(d)

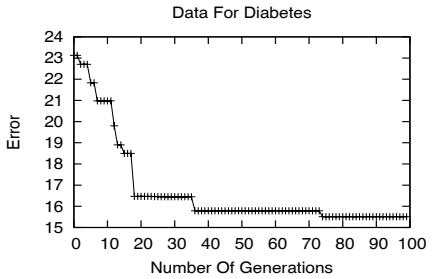
Fig. 5. Evolution of ANN's connection for four problems (a) breast cancer problem (b) heart disease problem (c) diabetes problem (d)thyroid problem



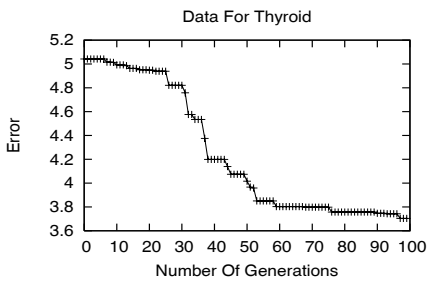
(a)



(b)



(c)



(d)

Fig. 6. Evolution of ANN's error for four problems (a) breast cancer problem (b) heart disease problem (c) diabetes problem (d)thyroid problem

Table 3. Architecture and Accuracy of Evolved Neural Networks

		No. of connections	No. of hidden nodes	No. of generations	Validation Set Error	Validation Set Error rate	Test Set Error	Test Set Error rate
Breast Cancer Data Set	Mean	93.9	11.78	28.56	2.3930	3.4057	2.4451	3.3678
	SD	15.32	1.84	17.85	0.1076	0.6460	0.3682	0.8368
	Min	6	9	11	2.0782	2.2857	1.7539	1.1494
	Max	119	16	114	2.4991	4.5714	3.2381	5.1724
Heart Disease Data Set	Mean	929.98	34.28	64.56	13.4517	17.8421	5.0392	5.4399
	SD	168.17	6.02	20.59	0.0476	1.2934	0.4820	1.5740
	Min	698	26	33	13.3033	15.7895	4.1711	2.6667
	Max	1284	51	117	13.4996	21.0526	6.5164	9.3333
Diabetes Data Set	Mean	78	13.06	62.36	15.8284	24.4479	17.7376	28.4374
	SD	25.49	3.37	48.06	0.1040	1.8114	0.9997	2.8298
	Min	24	6	19	15.5802	20.8333	15.6096	19.2708
	Max	150	21	292	15.9944	28.1250	20.3054	35.4167
Thyroid Data Set	Mean	465.61	23.5	35.67	4.4709	7.9014	3.9806	7.0390
	SD	89.45	5.29	3.08	0.0145	0.1135	0.0191	0.1217
	Min	282	14	32	4.4546	7.7106	3.9565	6.8214
	Max	655	29	41	4.4937	8.0286	4.0201	7.2345

3.4 Comparison with other Works

Direct comparison with other evolutionary approaches of designing ANN is very difficult due to the lack of such results. Instead, the best and latest results available, regardless of whether the algorithm used was an evolutionary, a BP or a statistical one, are used in the comparison. However, the aim of this paper is not just to compare this algorithm with all other algorithms but to establish a permutation free encoding scheme that could be used in existing algorithms. This section compares experimental results of PETENN with those of FNNCA [8], Prechelt's hand designed best NN (HDANNs) [3], EPNet [2], Bennet and Mangasarian's [9] MSM1, Schiffmann [10] and acasper [11].

Table 4 shows that PETENN achieved same result as of a manually designed ANN, which is better than that of FNNCA for breast cancer problem. EPNet has an absolute 0.00% error rate, which is presumably hard to gain for all real-world problems. One reason for the similar performance between PETENN and the manually designed best NN (HDANNs) is that the cancer problem is a relatively easy problem. Table 5 shows that PETENN has the lowest testing error rate among all other algorithms for the heart disease problem. It outperforms the rest by quite a significant margin. Table 6 shows that PETENN has earned testing accuracy better than that by Prechelt and same as EPNet for diabetes

Table 4. Comparing PETENN with other works for the breast cancer problem

	PETENN	FNNCA	HDANNs	EPNet
Testing Error Rate	0.01149	0.0145	0.0115	0.0

Table 5. Comparing PETENN with other works for the heart disease problem

	PETENN	MSM1	HDANNs	EPNet	acasper
Testing Error Rate	0.02667	0.1653	0.1478	0.1323	0.1696

Table 6. Comparing PETENN with other works for the diabetes problem

	PETENN	HDANNs	EPNet	acasper
Testing Error Rate	0.192708	0.2135	0.19271	0.2031

Table 7. Comparing PETENN with other works for the thyroid problem

	PETENN	Result by Schiffmann	HDANNs	EPNet
Testing Error Rate	0.068	0.025	0.0128	0.0163

problem. According to Table 7, PETENN performs a little bit worse compared to other approaches due to its failure to exploit the large training dataset.

4 Conclusion

The main contribution of this paper is to explore the genetic approach for evolving ANNs which surpassed the limitations of previous GA-based approaches. The representation of the networks and the incorporated sorting technique are innovative ideas that can solve the permutation problem. A mixed recombination operator has also been used. Another feature is the automation of addition or deletion of nodes or links during the recombination process.

PETENN has been tested on a number of benchmark problems that produced very competitive results compared to other algorithms. PETENN imposes very few constraints on feasible ANN architectures; thus faces a huge search space of different ANNs. It can escape from structural local minima due to its global search capability. The experimental results have shown that PETENN can explore the ANN space effectively.

As a future work, we plan to develop a similar encoding technique for multiple hidden layers. This will give the encoding technique a new dimension.

References

1. Das, A., Islam, R., Abdullah, S.M.: Evolving neural networks using evolutionary algorithm. Undergraduate Thesis, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (2007)
2. Yao, X., Liu, Y.: A new evolutionary system for evolving artificial neural networks. IEEE Transactions on Neural Networks 8(3), 694–713 (1997)

3. Prechelt, L.: Proben1-A set of neural network benchmark problems and benchmarking rules, Fakultat fur Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Technical Report 21/94 (1994)
4. Belew, R.K., McInerney, J., Schraudolph, N.N.: Evolving networks Using the genetic algorithm with connectionist learning. In: Langton, C.G., Taylor, C., Farmer, J.D., Rasmussen, S. (eds.) Workshop on Artificial Life, pp. 511–547. Addison-Wesley, CA (1992)
5. Hancock, P.J.B.: Genetic algorithms and permutation problems-A comparison of recombination operators for neural net structure specification. In: Whitley, D., Schaffer, J.D. (eds.) COGANN Workshop, pp. 108–122. IEEE press, Baltimore (1992)
6. Cybenko, G.: Approximation by superpositions of sigmoids. Mathematics of Control, Signals and Systems 2, 303–314 (1989)
7. Funahashi, K.: On the approximate realization of continuous mappings by neural networks. Neural Networks 2(3), 183–192 (1989)
8. Setiono, R., Hui, L.C.K.: Use of a quasinewton method in a feed forward neural-network construction algorithm. IEEE Transactions on Neural Networks 6, 273–277 (1995)
9. Bennett, K.P., Mangasarian, O.L.: Robust linear programming discrimination of two linearly inseparable sets. Optimization Methods Software 1, 23–34 (1992)
10. Schiffmann, W., Joost, M., Werner, R.: Synthesis and performance analysis of multilayer neural network architectures. Univ. Koblenz, Inst. fur Physics, Koblenz, Germany, Technical Report 16/1992 (1992)
11. Treadgold, N.K., Gedeon, T.D.: Exploring constructive cascade networks. IEEE Transactions on Neural Networks 10, 1335–1350 (1999)