Neuroevolution: Randomness is the Simplest Thing?[*]

Yury Tsoy
yurytsoy@gmail.com
Comp. Eng. Dept., Tomsk Polytechnic Univ.
Dept. of Economic Math., Informatics and Statistics,
Tomsk State Univ. of Control Systems and Radioelectronics
Tomsk, Russia

June 26, 2012

Abstract

The paper describes results of the research of neuroevolutionary (NE) algorithm with random scheme for selection of operators, which modify weights and structure of evolving artificial neural networks (ANNs). Not looking at the simplicity of the algorithm some already known heuristics for improvement of efficiency of the NE algorithms are observed in result of the algorithm analysis, for example: prevalence of weights mutation over other operators; decrease of mutation probability over time; advantage of using mutation of activation functions. Interestingly that some novel heuristics are found as well: denial to use node removal operation; adaptation of connections addition/removal depending on the initial ANN size. It was also found that a limited growth of the number of nodes and connections is inherent to neuroevolutionary algorithm and is quadratic in most cases. One of the most unexpected outcomes of the research is that analysis of usage of different mutation operators lead to an observation that many non-connected problems share similar properties in the sense of application of operator A after operator B, which gives strong hopes towards creation of algorithms, which can transfer knowledge while solving different problems. On top of that the algorithm was able to solve successfully several known benchmark problems, including the Artificial Ant problem (with 20% success rate), which is quite surprising for such a simple algorithm.

1 Introduction

When developing neuroevolutionary (NE) algorithms researchers often aim to make as strong algorithm as possible from the very beginning. To achieve this different heuristics can be introduced based upon researchers experience, observations, intuition and *ad-hoc* rules. A wide field for experimentation with

[*]Tsoy Y. Neuroevolution: Randomness is the Simplest Thing? ANRG Report No. 1201. Tomsk Polytechnic University, 2012.

such heuristics is the set of operators, which change structure and/or weights of artificial neural network (ANN). Different parameters for operators can be heuristically tuned like probability change over time, possible restrictions and dependencies on current ANN structure. Traditionally more attention is paid towards operators which involve growth of ANN structure complexity (growth of the components number). The examples are:

- Setting low probability of a new node arrival. This is often done to prevent fast growth of the number of nodes, because it brings multiple disadvantages, starting from making the ANN structure look more "complex", and ending with growth of the problem dimensionality, since the more the number of nodes, the more a size of the search space of neural structures.
- Way of adding a new node. There are to widely used options: (a) add a node with a couple of connections to make an immediate impact on ANN functioning [27, 42]; and (b) insert a node into some connection [38]. Note that these options do not consider adding isolated node, which is a pure result of the 'Add Node' operation, but prefer sort of 'macro' operation, which adds a node **and** connections into existing network.

Alike heuristics are basically intended to take into consideration a lot of factors connected with: operators, which change ANN structure; evolutionary search; problem at hand and its constraints. Nevertheless several successful NE algorithms were developed ([27, 38, 36]), which got a broad range of interesting theoretical and practical applications.

However in order to fully understand possibilities of modern NE algorithms it's necessary to find out what the simplest NE algorithm is capable for, to take it as a reference. For now a lot of researches made, dedicated to study of efficiency of evolutionary search for optimization of only neural weights [14] or structure [22]. But no research is made involving simplest version of the algorithm for simultaneous flexible evolution of structure and weights. Here 'flexible' means possibility of obtaining any neural network starting from any other network if these networks have the same set of inputs and outputs.

This paper concerns basic variant of NE algorithm with flexible search of neural structures. The algorithm doesn't use crossing and makes use of several operators: add node, remove node, add connection, remove connection, modify weight, modify activation. The operators, which change individual's ANN are picked at random and their acting is totally random. Surprisingly that even this variant of neuroevolutionary algorithm can provide very interesting observations like limited growth of neural structures' complexity, and possibility to share experience when solving different problems. And besides it'll be shown that even such a simple algorithm is able to solve several traditional benchmark problems of adaptive control and classification.

One more important and interesting question is creation of algorithms, which can learn from their experience. Most current approaches when solving the same problem multiple times will perform each run without knowledge of previous results. That is such algorithm can be said to have total *amnesia*. But all living beings perform in different way. Solving one and the same task for many times will eventually lead to improve of skills and knowledge and when faced similar by novel problem these skills can be helpful.

There are a lot of approaches, among which is incremental learning [9, 11, 14, 16], but these approaches mostly consider gradual complexification/change of the same problem or problem-dependent data over time, which means that the solution is able to operate in dynamic environments, but only if these environments are known to this solution. Ultimate goal is to develop a meta-learning approach, which will tell the learning algorithm how to learn solutions to new problems on the basis of obtained results and past experience.

In this paper we also address this question of meta-learning and present some results, which show that solving several completely different problem domains can lead to similar data for meta-learning algorithm. It enables creation of the algorithm, which can look for solutions in totally unknown environment using only its past experience without heuristics.

The paper is organized as follows. Sec. 2 gives a brief review on heuristics for known NE algorithms to direct evolutionary search. Sec. 3 contains description of the algorithm and operators to change ANN structure and weights, which are used for experiments. The experimental problems and parameters settings are given in the Sec. 4, while Sec. 5 contains results of experiments and their discussion. Conclusion outlines main results of the research and pays some attention to the future research.

2 Brief Review of Heuristics for Neuroevolutionary Algorithms

Below a brief review of some known heuristics to improve evolutionary search and training of ANNs.

All heuristics can be decomposed into two classes:

1. Heuristics for finding weights of ANN.
2. Heuristics for finding structure of ANN.

One of well-known general heuristic for weights forces absolute values of weights to lie within a small interval (e.g. $[-0.1, 0.1]$). This heuristics comes from a general theory of neural networks, where low valued weights are often preferred to avoid nodes from working in "saturated" regimes [17]. Those regimes make nodes fire only values close to minimal and/or maximal possible, without any intermediate states, which makes the whole ANN either irresponsible to input data changes, or otherwise very sensible (and thus unstable) to slightest changes.

The other heuristic for weights employs the idea of decomposition of initial problem into lesser subproblems. Each of these subproblems is solved separately, for example, using coevolutionary approach [14].

Heuristics for structures of ANNs are met more often, probably, because search of neural topologies is more sophisticated problem and there are major differences from numerical optimization. First of all the space of structures is non-differentiable and it's generally hard to define the direction of the "best" change. This direction should tell what changes need to be applied to current ANN to improve its performance.

To address the problem of search of neural networks structures various restrictions are introduced. For example, only multi-layered ANNs can be considered [31, 10, 41] and operations, which change its structure include adding or removal of a new layer or nodes for a selected layer. Considering NE algorithms with more flexible search of ANN topology, a popular trick is to allow only growth of ANN structure [38, 37]. In other words new elements (nodes and connections) are only added to the network, and there is no operator, which removes them. Some algorithms restrict growth by allowing only feed-forward ANNs [28, 42, 41], which also limits range of problems, which can be solved using obtained neural networks. In some algorithms the number of hidden nodes and connections is limited from above to make sure that ANN size won't exceed this limit [2, 1, 10, 22, 25, 28], or as a variant the number of connections per each unit can be fixed [27, 28].

For more precise guidance of the structure search, probabilities of different operators are adjusted. It's general to prevent too fast growth of number of hidden nodes and connections. Typical choice is setting probability of adding a new node much smaller than that of for adding a new connection or changing connection weight [38]. In [20, 42, 34] different operators' probabilities adapt on the basis of their relative 'quality', which is proportional to the fitness gain obtained due to this operator over the last several generations. In [42] operators' probabilities adapt to the structure of the underlying ANN, so that for ANN without hidden nodes a probability to add a new node is higher, than, for example, for ANN with 10 hidden units. Operators' probabilities can also be defined adaptively for each individual on the basis of the relative fitness of this individual in the population [2, 30].

One interesting variant to guide search of ANN structure is aimed towards adopting properties of real world complex networks, among which is a well-known exponential distribution of nodes degrees, which is typical for scale-free networks [4]. This property can be used like follows: when a new connection is added, its beginning and ending nodes are defined based upon degrees of nodes, so that nodes with higher degrees would have a larger probability to get this connection [44]. Deletion of connections can be performed in vice versa way: nodes with lower degrees will have higher probability to "loose" a connection. Selective removal of a connection can be alternatively implemented using the concept significance of deviation of the connection weight from zero [46].

If no restrictions on ANN structures are made, then appearance of "ugly" topologies is possible, alike the ones shown in the Fig. 1. The reason for this topologies to be acknowledged unsuitable is twofold: (1) there are elements, which do not affect neural output, however they participate in the computations performed by a network and thus increase overall computation complexity; (2) unbalanced topologies are more tricky things, however they are unlikely to be desired. In a few words, when we have chain of nodes, we try to transform only one variable to extract as much information as possible. However the processing would be more efficient if we could use information from other processing steps and other variables, which requires shortcut connections and neural nodes working in parallel. Thus ANN, where only a few of possible connections are toggled on can be supposedly considered as network, which misses some information. The latter doesn't mean that chains of nodes are always bad, but it argues, that such chains are generally not as good as layered networks.

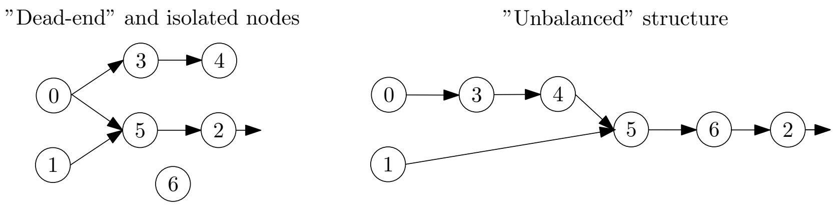

Figure 1: Example of bad structures, which can appear if no heuristic rules are applied to regulate neural topologies.

3 Algorithm Description

The algorithm scheme which is used for this research is standard for evolutionary populational algorithm except that no crossing is used and selected individuals immediately proceed to mutation step (Alg. 1).

Algorithm 1 Algorithm's scheme

```
1: Initialize population of $n$ ANNs without hidden nodes and with all inputs
   connected to all outputs.
2: Evaluate individuals.
3: Perform selection to pick $n$ individuals using Tournament Selection with a
   tour size = 5.
4: Mutate selected individuals. Individual's fitness is evaluated each time the
   mutation occurs to compute fitness change, implied by the mutation.
5: If stopping criterion is not satisfied then proceed to step 3.
```

The selective pressure was intentionally set high since this is a common recommendation to make evolutionary algorithm work well [8].

3.1 Encoding

Each individual contains information about weights and structure of ANN and types of activation functions for each non-input node (Fig. 2).

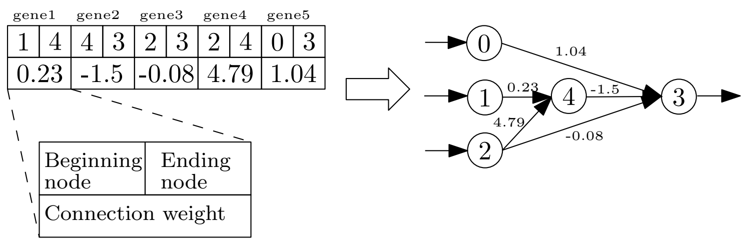

Figure 2: Neural networks encoding used in the paper. Each ANN is encoded directly by list of weighted connections.

Each node in ANN has its unique ID. Information about ANN structure is stored as list of edges, for each its start and finish nodes' IDs and weight are provided. List of activation functions is stored separately, possible entries are given in the Table 1. Default activation function is Sigmoid. All input nodes has Identity activations.

Activation	Formula
Identity	$f(x) = x$
Linear	$f(\mathbf{x}) = S$
Sigmoid	$f(\mathbf{x}) = \frac{1}{1+\exp(-aS)}$
Gaussian	$f(\mathbf{x}) = \exp\left(-\frac{\lVert\mathbf{x}-\mathbf{w}\rVert}{2a}\right)$

Table 1: Activations, which can be used during the ANN evolution. $S = \mathbf{w}^T\mathbf{x}$, where **w** – node's weight vector, **x** – input signals. $a \in \mathcal{R}$

3.2 Operators

The following operators are used in the study:

1. **Add Node**. Adds isolated node. This operations has no immediate effect on individual's fitness (neutral operation), but in some consequent generation this node can be advantageous. Note also that even adding connections to this node not obligatory lead to a fitness change, if there is no other node in the network that accepts output signal of this node.

2. **Remove Node**. Removes random node regardless on the number of its connections or feasibility of the ANN structure after this operation, that is without respect to whether input signals are translated onto ANN outputs or not. Input and output nodes can not be removed.

3. **Add Connection**. Adds connection with random weight between two randomly chosen nodes.

4. **Remove Connection**. Removes random connection from a network. If in result some node becomes isolated, then this node is removed ('dying').

5. **Modify Weight**. Changes weight of randomly chosen connection. The weight is changed using uniform distribution on interval $[w_{\min}, w_{\max}]$, where $w_{\min}$ and $w_{\max}$ – minimal and maximal allowed random weight values.

6. **Modify Activation**. Randomly changes activation function for randomly chosen node. Can not change activation for input nodes.

It's easy to see that these operators present baseline version of operators to change structure and parameters of ANN. All operators behave at random and doesn't adapt to the structure of ANN or some other characteristics (like fitness value, generation number and so on).

3.3 Operators Choice

Although quite a number of operators will be used, there is no scheme or heuristic, to decide which operator to pick at the moment. When some individual ANN undergoes mutation always a random operator is selected to act.

There is one feature though. For each individual $(N_I * N_O)$ mutation events are gambled, each firing with P_m probability, where N_I and N_O – are number of inputs and outputs of ANN correspondingly, and P_m is a mutation probability. This is done in order to follow "standard" bit-wise mutation concept: individual can mutate several times, dependent on the problem's size. Note that $(N_I * N_O)$ is exactly the number of connections in the initial ANNs, but at later evolution steps number of connections can change, however this coefficient doesn't adapt.

It is worth noting, that no restrictions is set to limit the operators acting, thus any kind of ANN topology (feed-forward or recurrent) can evolve, both growing and shrinking networks are considered and the number of nodes and connections can be arbitrary.

4 Experiments Description

The main goal of the experiments is to define what kind of new knowledge can be extracted from analysis of one of the simplest versions of the NE algorithm and how this information can be used. The other objective is to find out whether discovery of already known heuristics for evolution of neural structures can be made on the basis of analysis of the algorithm with random operators selection scheme.

To find this out, we'll track average number of nodes and connections in evolving neural networks. For more detailed analysis of operators the algorithm also writes log of changes that happened to the best found solution starting from the very 1st generation. It is assumed that list of operators, that caused those changes, carries important information on relative utility of each operator and probably on the possible successful sequences ('macro-operations') of those operators. Each entry in the list of changes will store:

- Name of the operator.
- Change of fitness, caused by the operator.
- Generation, at which the entry was created.

Having list of changes for the best individual the following parameters will be analyzed:

- Min, max, mean and variance of fitness change, caused by the operator.
- Usage of the operator over time (was the operator useful at the beginning only or it was used during the whole run?).
- Rate of successful transitions from operator 1 to operator 2 to understand if some information on the useful sequences of operators can be obtained.

Note that the analysis of the best individual can be noisy comparing to the analysis of each individual in the population because depends much on the initialization, random factors etc. However since EA converges to one solution (unless some special methods for maintaining diversity are applied) it can be suggested that in the final generation all the population will contain very similar individuals with very subtle differences in changes lists. This can be overcame by implementing Island Model for EA [12] or by adding niching schemes [19] or other techniques to encourage diversity, but is subject to a whole new research.

Ultimately when aiming at creation of the algorithms, which are able learn from their experience, the desired feature is ability to learn from a single run. That is if the algorithm manages to find small, but useful information how to perform better in the future from a single experiment, then this could meant that the algorithm has better abilities to learn. In many cases making repeating experiments can be costly or undesired for some reason thus this ability to learn using a small amount of data can be very powerful. It's also remarkable, that one of the current problems in machine learning concerns learning using small training set, which underpins the actuality of extraction of new knowledge in 'data-limited' situations.

Moreover when talking about utility of different mutation operators a single run can be enough because the operators are used many times during this run, that is why it is still possible to gather informative data.

Below the problems are described, which were used for experimental research of the baseline variant of the NE algorithm. For each problem we'll specify number of inputs and outputs and describe them.

4.1 XOR

This is a classification problem where ANN is to be trained to implement logical eXclusive OR operation. The difficulty here is that the problem is not linearly separable and thus hidden nodes are required to solve this problem. However one hidden node is enough [38]. To solve this problem ANN had 2 binary inputs for each variable and 1 continuous output. The evolution duration is 10000 generations.

4.2 Artificial Ant

In this problem artificial ant is considered to be placed in a 37×37 toroidal world with eatable pellets placed in a special order (Santa Fe trail [23], Fig. 3). The ant is given 400 steps and thus can not explore the entire world and hence should behave "smarter". ANN to control the ant had 4 inputs and 3 outputs, all continuous, described in the Table 2. The evolution duration is 10000 generations.

4.3 2-Poles Balancing

This is a standard adaptive control benchmark problem, in which ANN should control a cart, moving along a rail, to prevent falling of 2 poles of different length and mass, which are positioned on the cart [40]. The poles can sway back and forth and if a pole's angle from vertical position exceeds some threshold the task is considered failed. The 2-Poles Balancing problem requires successful control for 100k simulated time steps. Inputs and outputs description for this problem is given in the Table 3. The evolution duration is 10000 generations.

Input/Output	Description
Input #1	Is there a food ahead? (yes/no)
Input #2	Output #1 at previous time step
Input #3	Output #2 at previous time step
Input #4	Output #3 at previous time step
Output #1	Action "Move Forward"
Output #2	Action "Turn Left"
Output #3	Action "Turn Right"

Table 2: Description of neural inputs and outputs for the Artificial Ant problem

Input/Output	Description
Input #1	Cart's position
Input #2	Cart's velocity
Input #3	1st pole angle
Input #4	1st pole angular velocity
Input #5	2nd pole angle
Input #6	2nd pole angular velocity
Output #1	Controlling action with continuous value to move the cart

Table 3: Description of neural inputs and outputs for the 2-Poles Balancing problem

4.4 Proben1 problems

The Proben1 benchmark problems set was proposed by L. Prechelt [32]. This set includes several classification and approximation problems most of which initially taken from the UCI repository database [29] and is created to provide more appropriate comparison of different classification and approximation methods. This is obtained due to preprocessing of all the data made by Prechelt and by explicit division of data into training, validation and test sets, so that any algorithm trained on some data set would use the same training samples and would be tested on the same test set. Thus any algorithm is working in the same conditions and the only thing to compare is an algorithm's accuracy. Table 4 contains some information on the used problems. The evolution duration is 1000 generations.

5 Results of Experiments

5.1 Growth of ANN structure

The first important questions are how and how fast the number of nodes and connections changes when no heuristics or restrictions to neural structure are applied. Change of these parameters over time for different problems is shown in Fig.4 and 5.

The plots show that growth of the number of nodes and connections is rather small, even though it is not restricted from above and probability to add a new node was the same as that of other mutation operators. Note that

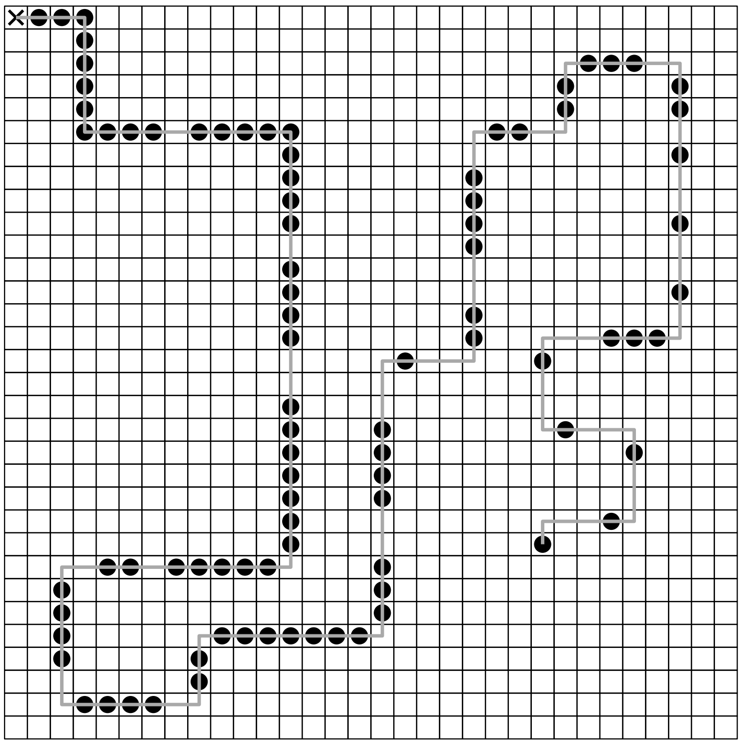

Figure 3: Visualization of the ant world known as "The Santa Fe Trail". Cross depicts initial position of the ant, food pellets are shown by black circles and optimal path is plotted by grey line.

at most cases the growth is almost quadratic. It's remarkable that coefficients at 3rd and 4th order variables for polynomial fitting equations are very small and even the 2nd order coefficients are less than 0.001. This allows to propose that ANN growth may be not a major problem, due to inherent self-restriction of the algorithm. Some possible causes of this self-restriction feature are discussed in the subsequent sections.

Sometimes a number of connections grows faster than that of nodes, which is rather natural, because when a new node is added the number of potential connections in the ANN grows proportionally to the square of the overall number of nodes. Sometimes the contrary was observed: the number of nodes grown faster then that of connections. Since the nodes are added without connections, it can be assumed that there were a lot of isolated ('silent') nodes, which did not not participate in ANN functioning, but could be used in the future. This circumstance (neutrality of the node addition) makes the probability to add a new node rather high, since this operation doesn't damage current ANN, but the situation can change if the node is added with connections with non-zero weights, i.e. if different operator for adding a node is used.

For some problems a number of connections reduces at the beginning, while for the other problems this parameter growths. Reduction of the connections number at the start of a run is observed for the problems with large initial number of connections, thus it seems that if the number of parameters is large, the

Problem's Name	Inputs	Classes	Training/Validation/Test Samples
cancer1	9	2	350/175/174
card1	51	2	345/173/172
diabetes1	8	2	384/192/192
glass1	9	6	107/54/53
heart1	35	2	460/230/230
horse1	58	3	182/91/91
thyroid1	21	3	3600/1800/1800

Table 4: Problems from the Proben1 set which are used in experiments. Number of inputs and outputs as well as samples in the training, validation and test sets are given for each problem.

algorithm tries to reduce it, without any 'built-in' heuristics! Hence the first found heuristic is *If the number of connections at the beginning is large, then try to remove connections. Otherwise, try to add a connection or node.*

5.2 Analysis of operators utility

We'll assume that operator is more *useful* if its application involves *better change* of fitness[1]. To estimate utility of operators, that change neural structure, the statistics for the changes induced by different operators was gathered for the best individual. Range plots, showing minimal, maximal and mean change of the fitness for different problems are shown in Fig. 6 and 7. The results were averaged over 10 runs.

The following notation is used: "+Node" – add node; "-Node" – remove node; "+Conn" – add connection; "-Conn" – remove connection; "Act" – change activation function; "Weight" – change connection weight.

Several interesting conclusions can be made from these plots. First of all the information on operators utility can differ significantly for different problems, which seems obvious. However in existing NE algorithms inherent heuristics concerning operators' usage do not change from one problem to another, although parameters setting may differ. This means that adaptive NE algorithms should be able vary heuristics by itself (self-adaptation) based upon their performance on the underlying problem.

Change of activation (underestimated by many researchers) can be very useful, which can be seen by large fitness change, observed after activation was changed, for many problems. For several problems mutation of activations for the best individual lead by chance only to positive changes (cancer1, card1, XOR). And there were problems, for which change of activation didn't produce any remarkable changes of fitness (see glass1 and Artificial Ant problems). Nevertheless, a heuristic can be formulated about *utility of mutation of activation*. It's worth noting that there are only few NE algorithms for flexible evolution of neural networks or analogous structures, which allow change of activation functions [27, 37, 44].

Probably the most intriguing observation concerns operator for removal

[1]Here "better" means larger for maximization problems and smaller for minimization problems.

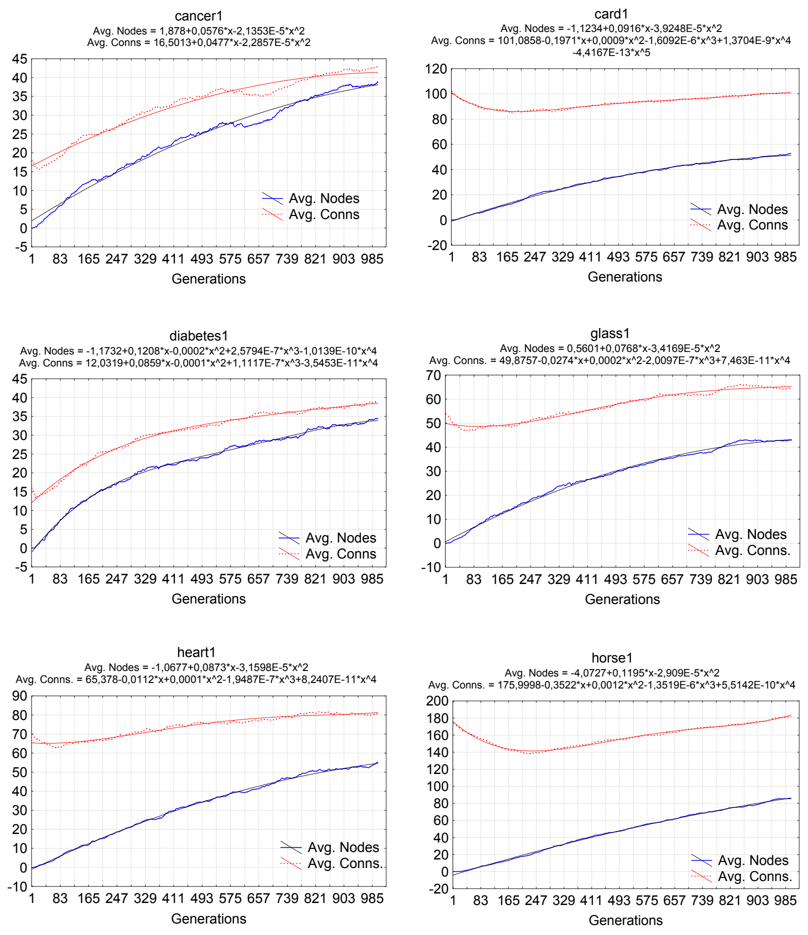

Figure 4: Change of average number of nodes and connections with fitting curves for problems from the Proben1 test set. On each plot time is shown along the horizontal axis and the number of nodes and connections – along vertical axis. Results are averaged over 10 runs.

of nodes. Interestingly that there are no stats for this operation, that is this operator *was never applied* to the best individual, although the number of calls of this operator was approximately the same as for other ones[2]. This fact is probably due to too large destruction, which is caused by node removal. This experimentally proves ignorance of the use of node removal operation, which can be seen in a popular algorithms like NEAT and HyperNEAT [38, 37]. Note, however, that these algorithms do not use this operator for another reason, namely, to consider only growing ANNs to reduce the size of a search space. Anyway the heuristic for *denial of node removal* is found here.

Although all these conclusions were obtained by analysis of the best individuals in population, they should hold in general because EA tend to converge to a single solution, unless special techniques for maintaining of diversity

[2]Due to equal chances for each operator to be chosen to perform mutation.

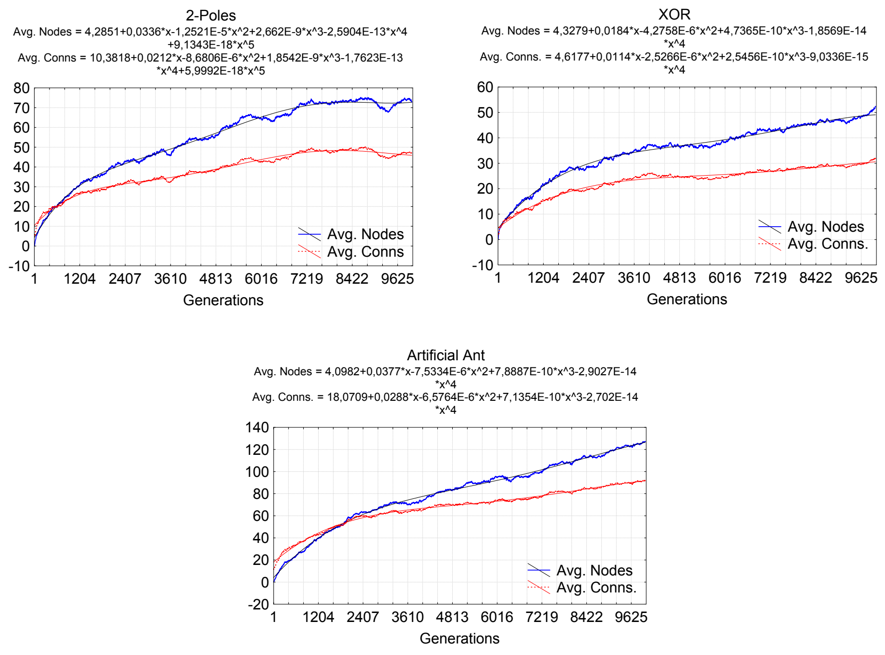

Figure 5: Change of average number of nodes and connections with fitting curves for XOR, 2-Poles Balancing and Artificial Ant problems. On each plot time is shown along the horizontal axis and the number of nodes and connections – along vertical axis. Results are averaged over 10 runs.

are applied.

5.3 Operators usage over time

Other interesting findings can be obtained via analysis of the number of usage of different operators at different stages of evolutionary search using the best individual statistics. That is when the algorithm stopped, the number of usages of each operator, ever applied to the best individual during the evolution was analyzed. Figures 8 and 9 show histograms for usage of operators during different periods of time.

It's clear that for many cases operator for mutation of weights prevails over others or at least among top-used operators. For some problems (card1, heart1 and horse1) this operator is used twice as often as the operator ranked second, while for other problems (cancer1, diabetes1, glass1) this advantage is not so large and tend to decrease over time. Anyway the *heuristic for more frequent usage of weights mutation* can be stated. This seems to be very natural because weights mutations allow fine tuning of ANN parameters and more detailed testing of existing topology, while changes of structure often lead to significant changes of ANN properties and performance.

Interesting observation is that more frequent use of weights mutation seems inherent to the solutions for classification problems, while it seems that for other problems structural changes are more important. The suggestion can be made, that for classification problems change of ANN weights leads to mi-

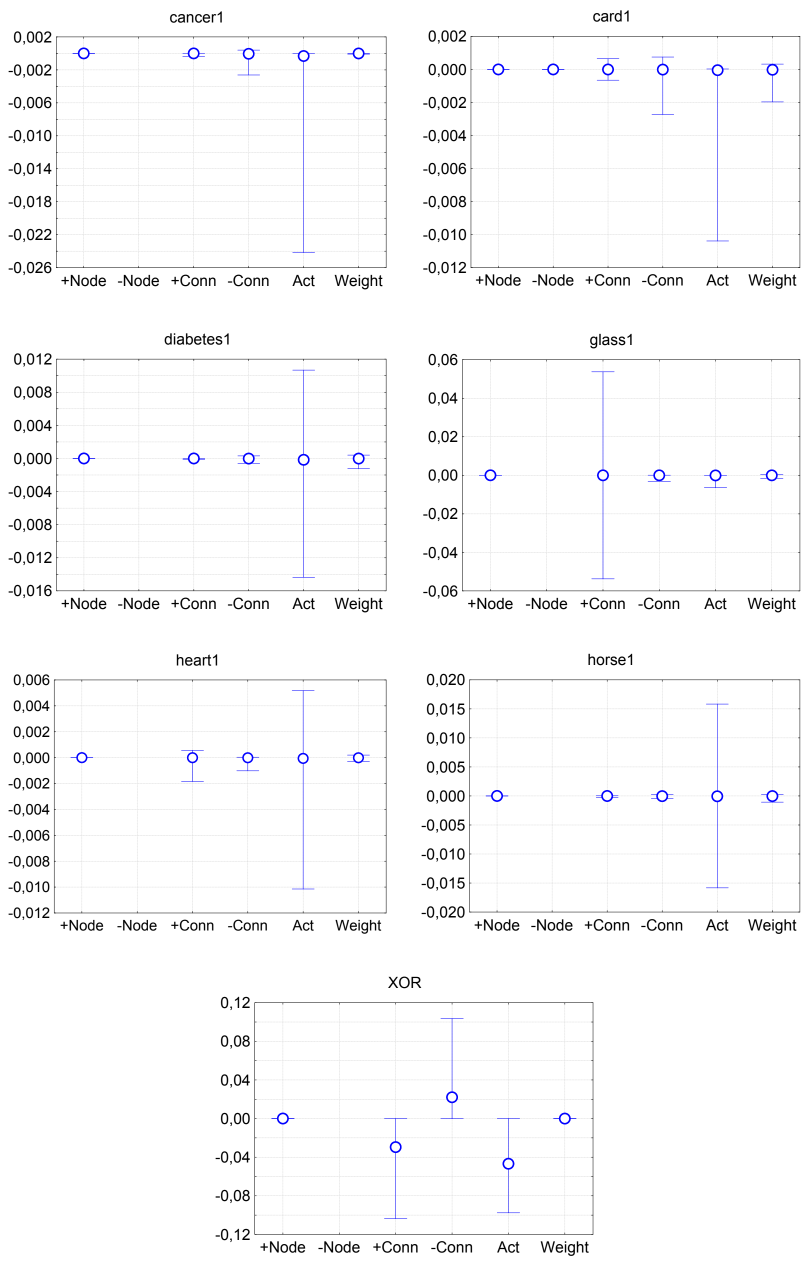

Figure 6: Utility of operators for minimization problems shown as averaged change of fitness, involved by application of correspondent operator. Minimal, maximal and mean fitness changes are plotted averaged over 10 runs.

nor changes of functionality (the number of correctly classified objects or mean squared error of ANN output tend to change slightly), while for control and adaptive behavior problems wrong action can ruin all other consequent actions. In other words, for the classification problems current ANN functioning doesn't depend on previous ANN responses, while for many control problems

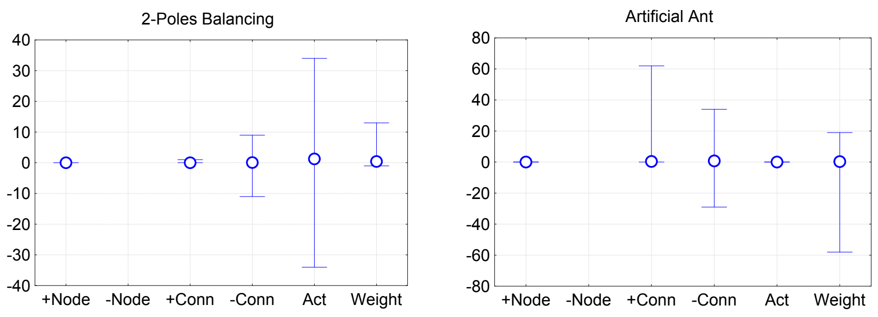

Figure 7: Utility of operators for maximization problems shown as averaged change of fitness, involved by application of correspondent operator. Minimal, maximal and mean fitness changes are plotted averaged over 10 runs.

this is crucial. For example in the Artificial Ant problem if the ant made a single wrong movement, then all other movements should be corrected, otherwise the ant will follow a wrong course.

For the XOR problem generation of the last improvement of the best individual was equal to 10. It means, that the best solution wasn't improved since then even though the runs lasted for 10000 generations. As a result the solution for this problem was not found and this means, that lack of improvements of the best individual indicates stagnation of a search, which was observed by many other researchers and lead to creation of the restarts strategy, see for example [3, 6, 21, 13, 35].

The other observation is that total number of mutations that lead either to improvements of the best solution or to slight fitness reduction tend to decrease over time. This decrease can be linear or exponential (see histograms for the 2-Poles Balancing or Artificial Ant problems). Related effect was observed many times in evolutionary algorithms, which lead to the heuristic for *reduction of the probability of mutation over time* [18, 3, 30]. Here it is shown that reduction of probability of a successful mutation is inherent to the neuroevolutionary search.

Note also that similar research for more sophisticated probabilistic scheme of operators selection was described in [20, 34]. Their results show that operators' utility varies over time depending on the state of the search process. It's remarkable that curves describing change of operator selection probability depend on the problem, which underpins necessity for adaptation of operators' probabilities. Experimental results show that node deletion has relatively low probability during almost the entire search, and this agrees with our finding of low utility of this operator. On the other hand weight mutation operator significance was lower than that of in our experiments. But in the research presented in [20, 34] different set of operators was considered and the analysis was performed for the entire population, not only for the best individual, so that poorly fitted individuals could also contribute to the obtained plots and graphs.

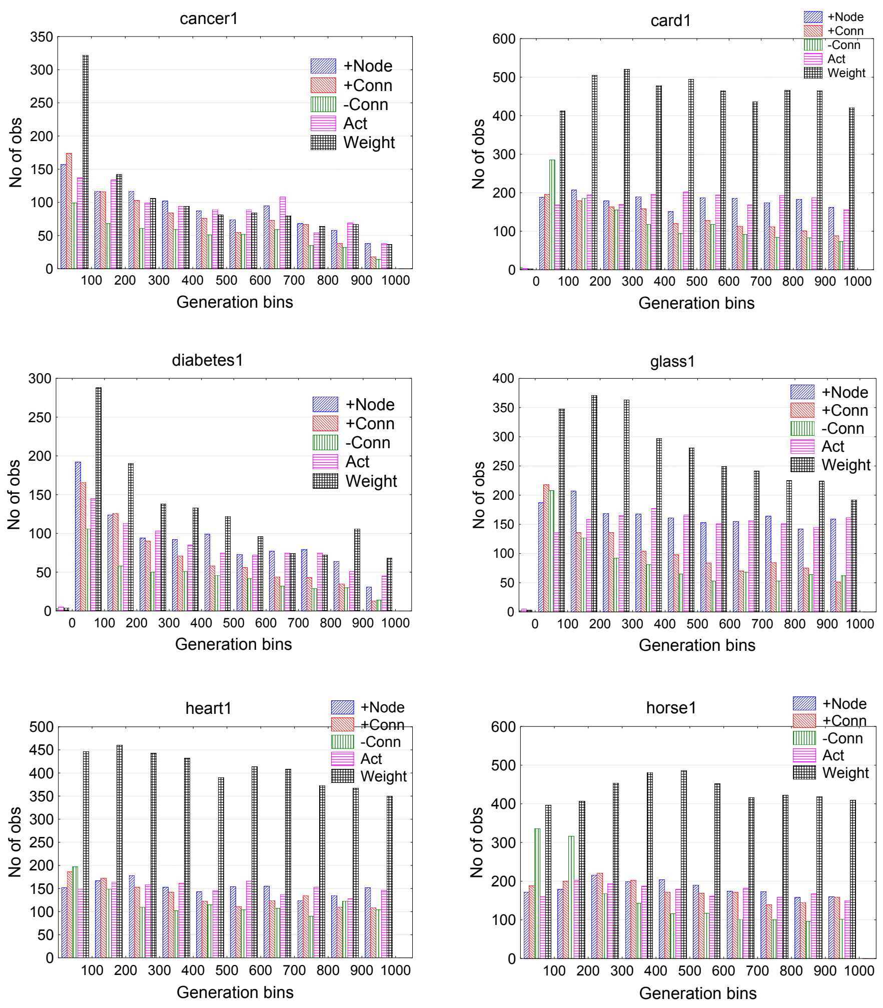

Figure 8: Usage of operators over time for the Proben1 problems. Operators notation from the previous figures applies. Results are averaged over 10 runs.

5.4 Analysis of Successful Operators Transitions

After understanding that number of usages of different operators and their utility vary over time it is interesting to find out whether sequences of operators are possible? In other words the traditional NE algorithm applies operators one-by-one without remembering, which operator was applied at the previous step. However selection plays the role of a filter, to choose which operators make successful improvements and which do not. The individuals, which were selected, do not have entirely random history of changes, due to selective pressure, thus a hypothesis can be made that there should be stable sequences of application of operators (analogous to high-level macroses or templates).

To check this hypothesis an analysis of successful application of operators was made with respect to operators, acted at the previous time step. For exam-

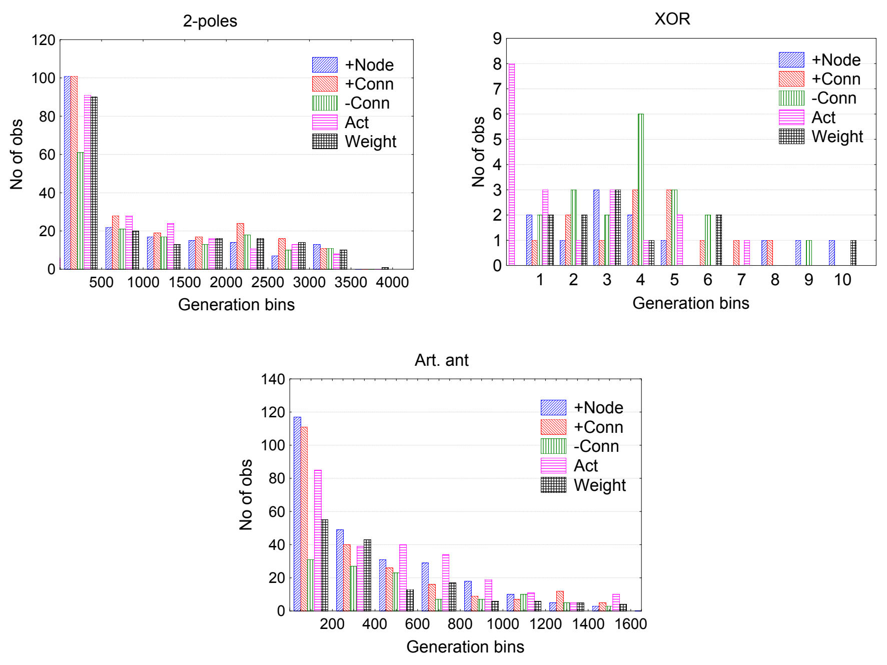

Figure 9: Usage of operators over time for the 2-Poles Balancing, XOR and Artificial Ant problems. Operators notation from the previous figures applies. Results are averaged over 10 runs.

ple, suppose that some individual has the following history of changes (minimization problem is considered):

Operator Name	Fitness change	Generation
+Node	0	2
Weight	$-6.73E-06$	3
Weight	$7.24E-06$	5
-Conn	-0.01	6
+Node	0	7
+Node	0	7
Act	0	7
Weights	$-1.28E-06$	7
+Conn	0	9
Weight	$-4.02E-05$	9
-Conn	$3.79E-05$	11

where the first column contains operator name, the second column shows fitness change implied by the operator and the last column contains a time label. It can be seen that there were advantageous (negative fitness change), harmful (positive fitness change) and neutral operations. From this history log we can extract sequences of operations of length 2 to judge their utility via average change of fitness and the number of successful outcomes. For example, the sequence

$$+\text{Node} \rightarrow \text{Weight}$$

is met only once with 1.0 probability of success, while

$$\text{Weight} \to -\text{Conn}$$

is met twice and was successful with probability 0.5.

Having much larger history of changes, obtained after hundreds and thousands of generations spend to solve some problem, it is possible to obtain more reliable statistics to build a more reliable matrix of successful transitions, which shows the probability of successful application of operator A if operator B acted the last time. Visualization of such matrices, showing averaged probabilities of successful transitions for each pair of operators for the best individuals, is shown on Fig. 10.

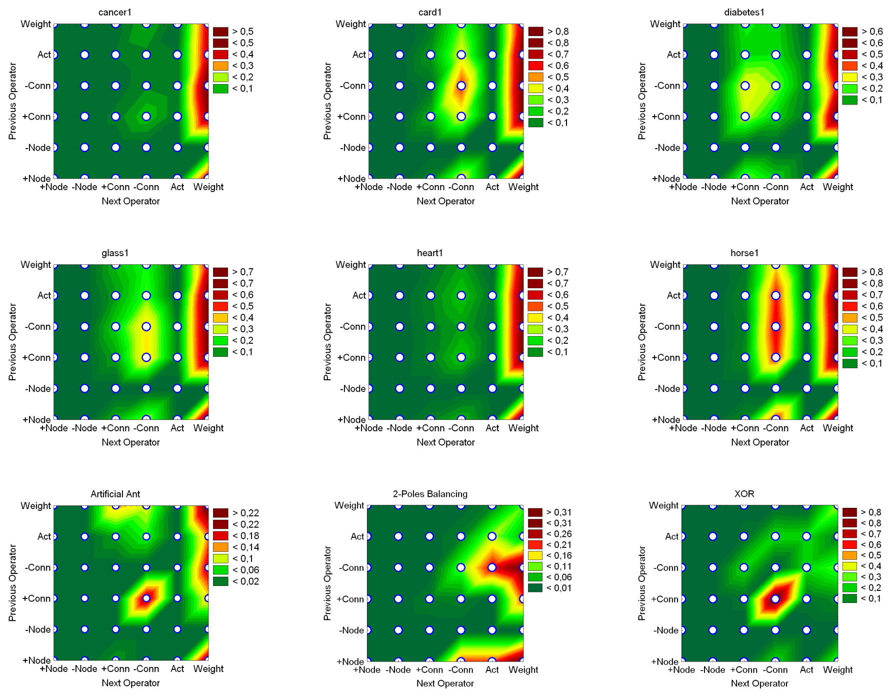

Figure 10: Visualization of transitions matrices for different problems. For each subpicture probability of successful transition are shown with respect to the previous operator, which modified ANN. Green color marks lower probability, and red marks higher probability. Scales for sub-pictures are independent. Results are averaged over 10 runs.

The most surprising result is that successful transitions have similar distributions even though the problems are different. In the majority of cases weights mutation has the largest probability of success. Absence of information for the node removal can be explained by the same fact as earlier: this operator wasn't used by a single best solution. From this observation one more heuristic can be proposed: *Solving of different problems by neuroevolutionary algorithm can have a lot in common*[3].

[3]Note that in the Subsec. 5.2 an observation was made that operators utility change from one

This heuristic encourages creation of NE algorithms with incremental and life-long learning and transfer of algorithm's past experience to novel problem domains. It's worth noting that some successful research in this direction is already made [14, 39, 45], but this area is obviously among the most unexplored, since there is no correspondent working algorithm or concept in any field of AI. This domain seems to be of great importance to create truly adaptive algorithms, which could adapt not only during their run, but also between runs by analyzing obtained performance to "mine" new possible regularities and rules.

One non obvious outcome from analysis of transitions matrices is that removal of connection was more successful than addition of connection. This could be one of the causes to hold growth of the network size, observed in the Section 5.1. Hence simplification of neural structures can be one more inherent property of NE algorithms.

There is one case, which stands aside. Transitions probability obtained by solving XOR problem is very different from the others. This can be rather easy explained by the fact, that for this problem evolution of the best solution stopped after 10th generation (see Fig. 9), thus too few information was available to produce more reliable statistics on successful probability of transitions. It also indicates that transition matrix can vary over time depending on the state of the evolutionary search.

To prove this resetting of parameters was made to increase mutation probability to encourage more active search of solutions. In results an ANN was evolved, which successfully solved the XOR problem. Its successful transitions probability matrix is visualized at Fig. 11. It is clear that the new matrix has similar traits to other matrices.

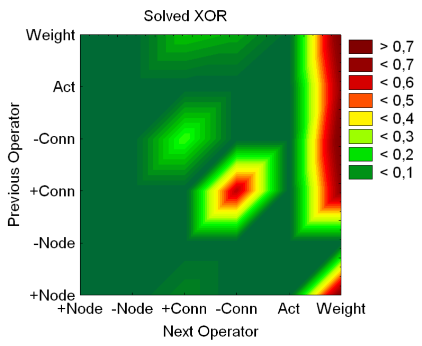

Figure 11: Visualization of successful transitions matrix for the ANN which eventually solved the XOR problem. Note similarities with most matrices at Fig. 10.

problem to another, which seems contradictory to the heuristic just stated. However, there's no contradiction, since large operator utility doesn't imply high usage rate and vice versa high probability of successful application of operator A after operator B doesn't tell anything about average change of fitness due to these operators.

5.4.1 Transferring experience between runs

Obtained experimental observations and findings lead to an extremely interesting question: if one got data on successful run of NE algorithm, can this data be applied to obtain many successful runs? In other words this question concerns possibility of self-learning of NE algorithm between runs.

To address this question we tried to use successful transitions matrix for XOR problem, shown in Fig. 11, to pick operators for other runs instead of purely random selection of operators. The algorithm for selection of operators is shown by Alg. 2.

Algorithm 2 Selection of operators using successful probability matrix

```
1: $\mathbf{T} \leftarrow$ matrix of successful transitions probabilities.
2: $Op_{-1} \leftarrow$ operator acted last
3: $\mathbf{v} \leftarrow$ vector of transitions probabilities extracted from $\mathbf{T}$ corresponding to $Op_{-1}$
4: $\mathbf{v} \leftarrow \mathbf{v} + 0.1$ {Increase each element in $\mathbf{v}$ by 0.1 to enable selection of operators with 0 probability.}
5: $\mathbf{v} \leftarrow (\sum_i \mathbf{v}_i)^{-1}\mathbf{v}$ {Normalize probabilities.}
6: $Op \leftarrow$ operator selected according to the distribution $\mathbf{v}$.
```

However such trials did not lead to success and behaved not better than all previous unsuccessful runs. Some probable reasons for this:

1. Different initialization of weights.
2. Transitions matrix is averaged over the whole search period and gives only the generalized probabilities of successful transitions. It is well-known that evolutionary algorithm run undergoes several stages [7, 33] and at these stages different properties are required, thus the algorithm should track applicability of the pre-defined matrix of successful transitions to decide whether it should be used. Moreover in [20] it was shown that for different problems different dynamics of operator probabilities should be employed.
3. This argument is inspired by the Bayesian decision making: there is a need to track operators utilities during the run ("online"). The rule for selection of operators should rely not only upon conditional probability of success, but also on some measure for applicability of previous operator, like frequency of successful actions in the past.

What is important here is not the inability to repeat good result using information from the previous successful run using some simple approach. More important is that this problem requires much deeper study and should not be taken easily.

5.5 Testing the algorithm

Even though the algorithm, which is used in this study wasn't designed to achieve maximal performance, but rather to study effects of randomness of operators selection, it is interesting to compare it with other algorithms. The results of solving classification problems from the Proben1 problems are given in the Table 5, the "heuristicless" NE algorithm under study is referred as NE_0.

Table 5: Comparison of the test set classification errors (%) for the Proben1 problems obtained using different approaches. ANN – results for "standard" training of ANN using RPROP algorithm [32]; GA – training of ANN weights using Genetic Algorithm [5]; pPCA – results for training of ANN with neuroevolutionary dimensionality reduction [43].

Problem	ANN	GA	pPCA	NE_0
cancer1 (9)	1.38	1.24	1.78	4.02
card1 (51)	14.05	14.27	15.64	14.77
diabetes1 (8)	24.10	23.70	25.00	30.63
glass1 (9)	32.7	47.62	32.07	53.23
heart1 (35)	19.72	21.87	20.00	21.26
horse1 (58)	29.19	26.44	30.66	27.36

The comparison results show that even the algorithm with the simplest scheme of operators selection can perform on a par with more sophisticated methods. For three problems, cancer1, diabetes and glass1, results of NE_0 were worse, however for cancer1 rather good result was obtained, and for diabetes1 and glass1 problems all other algorithms didn't show outstanding results either.

Artificial Ant problem was solved in 20% of runs, which means that the ANNs were found, which managed to gather all 89 pellets. This is rather surprising because this problem is considered hard in the genetic programming society [24, 26]. The worst result for found solutions in unsuccessful runs was 71 pellet.

2-Poles Balancing and XOR problems were hard nuts to crack for the NE_0 algorithm. Probably this could mean that these two problems can not be solved by "heuristicless" algorithm. A hypothesis can be made that in order to solve these problems successfully the algorithm should support "chains of improvement". That is if the ANN has a structure, with which the solution is impossible, then successful change might present a sequence of operations, which should not be broken by other mutations. This chain can potentially include operations that reduce the solution's performance or at least doesn't improve it. In the XOR problem ANN without hidden nodes can not be a solution, because at least one hidden unit is required with proper connections and weights. For the algorithm, which is studied in this paper, multiple mutations are necessary to construct a network, which could be elaborated to such a solution, and it takes several generations because at each generation only not more than 2 mutations take place per each individual (see Section 3.3 for details). So the chance for such an improvement is negligible and this can explain the failure to solve the XOR problem.

As for the 2-Poles Balancing problem a deeper study is required, but a suggestion can be made that obtaining a solution requires temporal deterioration of the current one, which is also discouraged by the algorithm under use. I.e. probably the same "chains of improvement" issue takes place. It is possible to guard new solutions even if they show worse performance, for example, by prohibiting other individuals to compete with them as it's done in the NEAT algorithm [38].

Anyway the fact that the algorithm with purely random selection of operators was able to show quite a modest performance is surprising by itself.

6 Conclusion

Not looking at the simplicity of the algorithm under study the following heuristics and observations were found, some of which are new and wasn't found in the literature:

1. If the number of connections at the beginning is large it's worth to remove some of them. Otherwise, the new connection or node should be added.
2. Mutation of activation function is an advantage.
3. Operator, which removes nodes, is at least not obligatory.
4. Weights mutation should be used more frequently than other operators.
5. It is shown that reduction of probability of a successful mutation is inherent to the neuroevolutionary algorithms.

The found heuristics show that basic variant of the NE algorithm with flexible search of ANN structures is quite "intelligent" by itself. Nevertheless there's always a room for improvement by adding more powerful heuristics, which will enforce the inherent properties.

Another remarkable observation concerns matrices for probabilities of successful application of operators (transitions matrices) obtained for different problems. Not looking at differences of those problems the matrices structures look very similar although variation in absolute values can be rather large. It is important that a transition matrix changes during the algorithm run and lack of statistical information leads to large differences in probabilities of operators success. A quick trial to use transition matrix from a successful run to produce more successful results failed, which clearly indicates that the problem of life-long learning of NE algorithm and reusability of the run outcomes requires much deeper study and should not be taken easily.

Another interesting observation concerns the Artificial Ant problem. This problem is considered hard, however it was successfully solved by a very simple neuroevolutionary algorithm, although the reliability was not large (20 %).

The crossover operator wasn't used in this research, although there are ways to cross neural networks with different structures [15, 22, 27, 38, 42, 44]. Use of crossover will most likely change the results and probably some conclusions and found heuristics, because frequency of operators usage and other theirs stats would obviously be dependent on crossover results. This can be an interesting task for a future research.

7 Acknowledgments

This research was partially supported by the Russian Foundation for Basic Researches (projects no. 11-07-00027-a and 12-08-00296-a).

References

[1] J. H. Ang, K. C. Tan, and A. Al-Mamun. Training neural networks for classification using growth probability-based evolution. *Neurocomputing*, 71(16-18):3493–3508, October 2008.

[2] P.J. Angeline, G.M. Saunders, and J.B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, 1993.

[3] A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005*, pages 1769–1776, 2005.

[4] Albert-Laszlo Barabasi. *Linked: How Everything Is Connected to Everything Else and What It Means*. Plume, 2003.

[5] F.Y. Barrera. Busqueda de la estructura optima de redes neurales con algoritmos geneticos y simulated annealing. verificacion con el benchmark proben1. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 11(34):41–61, 2007.

[6] Giuseppe Cuccu and Faustino Gomez. Novelty restarts for evolution strategies. In *Proceedings of the 2011 Congress on Evolutionary Computation (CEC 2011).*, 2011.

[7] Kenneth De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral dissertation, University of Michigan, Ann Arbor, 1975. University Microfilms No. 76-9381, 1975.

[8] Kalyanmoy Deb and Samir Agrawal. Understanding interactions among genetic algorithm parameters. In *Foundations of Genetic Algorithms 5*, pages 265–286. Morgan Kaufmann, 1999.

[9] R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22:1517–1531, 2011.

[10] D. B. Fogel. Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe. In *International Conference on Neural Networks*, pages 875–880, San Francisco, CA, 1993. IEEE Press.

[11] Glenn Fung and Olvi L. Mangasarian. Incremental support vector machine classification. In *Proceedings of the 7 th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2001.

[12] David E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the Third Internation Conference on Genetic Algorithms*, pages 70–79, 1989.

[13] Faustino Gomez. *Robust Non-Linear Control through Neuroevolution*. PhD thesis, Department of Computer Sciences, 2003.

[14] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[15] F. Gruau. *Neural network synthesis using cellular encoding and the genetic algorithm: Unpublished PhD thesis*. PhD thesis, lUniversite Claude Bernard, Lyon, 1994.

[16] S.U. Guan, Y. Qi, and C. Bao. An incremental approach to mse-based feature selection. *International Journal of Computational Intelligence and Applications*, 6(4):451–471, 2006.

[17] Simon Haykin. *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall, 1998.

[18] J. Hesser and R. Manner. Towards an optimal mutation probability for genetic algorithms. In Manner R. Schwefel, H.-P., editor, *Proceedings of the 1st Conference on Parallel Problem Solving from Nature. LNCS No. 496*, pages 23–32, Berlin: Springer-Verlag, 1990.

[19] J. Horn. *The Nature of Niching: Genetic Algorithms and the Evolution of Optimal Cooperative Populations*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[20] C. Igel and M. Kreutz. Operator adaptation in evolutionary computation and its application to structure optimization of neural networks. *Neurocomputing*, 55(1-2):347–361, 2003.

[21] Thomas Jansen. On the analysis of dynamic restart strategies for evolutionary algorithms. In *Parallel Problem Solving from Nature – PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 33–43, 2002.

[22] H. Kitano. Designing neural network using genetic algorithm with graph generation system. *Complex Systems*, 4:461–476, 1990.

[23] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[24] W. B. Langdon and R. Poli. Why ants are hard. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, 1998.

[25] F.H.F. Leung, H.K. Lam, S.H. Ling, and P.K.S. Tam. Tuning of the structure and parameters of neural network using an improved genetic algorithm. *IEEE Transactions on Neural Networks*, 14(1):79–88, January 2003.

[26] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2009.

[27] C. Mattiussi, P. Durr, D. Marbach, and D. Floreano. Beyond graphs: A new synthesis. *Journal of Computational Science*, 2(2):165–177, 2011.

[28] D.E. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

[29] D. Newman, S. Hettich, C. Blake, and C. Merz. Uci repository of machine learning databases, 1998.

[30] T.H. Oong and N.A. Isa. Adaptive evolutionary artificial neural networks for pattern classification. *IEEE Transactions on Neural Networks*, 22(11):1823–1836, October 2011.

[31] D.L. Prados. New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques. *Electronics Letters*, 28(16):1560–1561, July 1992.

[32] L. Prechelt. Proben1 - a set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Fakultat fur Informatik, Universitat Karlsruhe, Karlsruhe, Germany, 1994.

[33] Adam Prugel-Bennett and Alex Rogers. Modelling ga dynamics. In *Theoretical Aspects of Evolutionary Computing*, pages 59–86, 1999.

[34] Benjamin Roeschies and Christian Igel. Structure optimization of reservoir networks. *Logic Journal of the IGPL*, 18(5):635–669, 2010.

[35] Daniel Rothman, Sean Luke, and Keith Sullivan. Do multiple trials help univariate methods? In *Proceedings of the 2011 Congress of Evolutionary Computation*, pages 2391–2398. IEEE Press, 2011.

[36] Kenneth O. Stanley, David B. D'Ambrosio, and Gauci Jason. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[37] K.O. Stanley. Patterns without development, 2006. Technical report CS-TR-06-01, University of Central Florida, 2006.

[38] K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[39] K.O. Stanley and R. Miikkulainen. Evolving a roving eye for go. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2004)*. New York, NY: Springer-Verlag, 2004.

[40] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[41] Alexander P. Topchy, Er P. Topchy, Oleg A. Lebedko, and Victor V. Miagkikh. Fast learning in multilayered neural networks by means of hybrid evolutionary and gradient algorithms. In *Proc. of IC on Evolutionary Computation and Its Applications*, pages 390–399, 1996.

[42] Y.R. Tsoy and V.G. Spitsyn. Using genetic algorithm with adaptive mutation mechanism for neural networks design and training. *Optical memory and neural networks*, 13(4):225–232, 2004.

[43] Yury Tsoy. Evolving linear neural networks for features space dimensionality reduction. In *Proceedings of the 2012 IEEE International Joint Conference on Neural Networks (IJCNN 2012)*, 2012.

[44] Yury R. Tsoy. Computational regulatory networks and self-adaptive neuroevolutionary algorithm. In *11th Int. Conf. with Int. Participation on Artificial Intelligence (CAI-2008)*, volume 3, pages 50–57. Moscow: LENAND, 2008. In Russian.

[45] Phillip Verbancsics and Kenneth O. Stanley. Evolving static representations for task transfer. *Journal of Machine Learning Research*, 11:1737–1769, 2010.

[46] X. Yao and Y. Liu. Evolutionary artificial neural networks that learn and generalise well. In *Proc. of the 1996 IEEE International Conference on Nueural Networks (ICNN'96)*, volume on Plenary, Panel and Special Sessions, pages 159–164, 1996.