Environments Conducive to Evolution of Modularity

Vineet Khare, Bernhard Sendhoff, Xin Yao

2006

Preprint:

This is an accepted article published in International Conference on Parallel Problem Solving From Nature (PPSN). The final authenticated version is available online at: https://doi.org/[DOI not available]

Environments Conducive to Evolution of Modularity*

Vineet R. Khare[1], Bernhard Sendhoff[2], and Xin Yao[1]

[1] Natural Computation Group, School of Computer Science,
The University of Birmingham, Birmingham B15 2TT, UK
{V.R.Khare, X.Yao}@cs.bham.ac.uk
WWW home page:http://www.cs.bham.ac.uk/research/NC/
[2] Honda Research Institute Europe GmbH, Carl-Legien-Straße 30,
63073 Offenbach/Main, Germany
Bernhard_Sendhoff@de.hrdeu.com
WWW home page:http://www.honda-ri.de

Abstract. Modularity has been recognised as one of the crucial aspects of natural complex systems. Since these are results of evolution, it has been argued that modular systems must have selective advantages over their monolithic counterparts. Simulation results with artificial neuro-evolutionary complex systems, however, are indecisive in this regard. It has been shown that advantages of modularity, if judged on a static task, in these systems are very much dependent on various factors involved in the training of these systems. We present a couple of dynamic environments and argue that environments like these might be partly responsible for the evolution of modular systems. These environments allow for a better, more direct use of structural information present within modular systems hence limit the influence of other factors. We support these arguments with the help of a co-evolutionary model and a fitness measure based on system performance in these dynamic environments.

1 Introduction

There are modular systems all around us in nature. The most cited example is the human brain, which is modular on several levels [1]. Macroscopically, we can observe specialised areas for certain tasks, like for visual (V1-V2-V4-ITL (inferior temporal lobe)) or auditory processing. On a mesoscopic level the structural re-occurring element is the column, and even on the microscopic level neurons can be grouped into structurally distinct classes, e.g. pyramidal neurons. This structural organisation is a results of evolution by natural selection. Often debated are the reasons for evolution of these modular systems. Models that try to explain the evolution of modularity can be divided into two categories [2]. Models in the first category have direct relationship between modularity and selective advantage. In this category some of the reasons presented for the evolution of

* This work is partly funded by Honda Research Institute Europe GmbH.

modularity are evolvability, phenotypic robustness against environmental perturbations, ease of learning etc. The other category includes model that do not assume a direct relationship but explain modularity as a side effect of dynamics of evolution.

This paper is concerned with models that relate the selective advantage of modularity with the effectiveness of learning in individuals. Hence these models deal with interaction between genetic modularity and learning. It has been argued [3] that during their lifetimes individuals need to learn more than one tasks simultaneously and having a modular structure helps in avoiding conflicting messages from these tasks. In Artificial Neural Network (ANN) literature this has been shown with the help of modular neural networks (MNNs) - as they outperform fully-connected structures. The classic example being the "what" and "where" vision tasks. Attempts have also been made to illustrate that this advantage of MNNs can lead to their evolution, using neuro-evolutionary methods. In [3] it was shown that modularity can be evolved on the basis of this advantage. In this work an ANN's architecture was genetically determined and evolved by mutation and selection, while its fitness was dependent on how well it performs on these "what" and "where" vision tasks, after training with backpropagation. This modular (non) interference based advantage, however, is not universal and depends on various factors involved in the training of the network. In [4] it was shown that this advantage depends crucially on the choice of cost function used for training and a proper choice can lead to superior non-modular structures. Our earlier experiments show that this also depends on the choice of mode of training (batch or incremental) and learning algorithm [5]. These results indicate that (a) it is possible to deal with modular interference with non-modular structures and (b) either the tasks and simulations considered are too simplistic to extract the benefits of modularity in complex systems or, simply, learning efficacy is not the reason behind the evolution of modularity.

For this work we take a different approach and consider a couple of dynamic environments, unlike the static environments considered in these models. We show that because of the nature of tasks in these environments the structural information within a MNN can be exploited more directly (Sect. 2) and hence limits the effect of other parameters. We then use a co-evolutionary model (Sect. 3) to show that modularity can be evolved in conditions earlier shown to be unappreciative towards the evolution of modularity. We discuss various experiments and results in Sect. 4 and finally conclude in Sect. 5. We would like to emphasize at this point that the aim of this work is not to design optimal systems for these dynamic environments, but to use these environments to contribute to the understanding of evolution of modularity in nature.

2 Dynamic Environments

Here we consider two different kinds of environments. In the first one an individual is not just expected to learn a given task but is also expected, afterwards, to adapt to a related task, which shares some of its characteristics with the origi-

nal task. In the other scenario an individual is expected to learn more and more complex task incrementally. While adapting to a related task the individual is expected to learn a function of form $g(f_1, f_2)$ and then adapt to $g'(f_1, f_2)$ and while adapting to a more complex task a function is expected to learn $g(f_1, f_2, f_3)$ after learning $g(f_1, f_2)$. Let us assume a MNN with matching topology, after learning the first function in phase-one, is required to adapt to the second function (Fig. 1) in phase-two. For incrementally complex tasks, individuals are allowed to grow. In this section we look at how modular and non-modular systems adapt in these environments. As examples of these we consider boolean functions (see Sect. 2.1 and 2.2). The goal here is not to solve these simple boolean problems, but to use these to gain a deeper understanding of when modularity is useful.

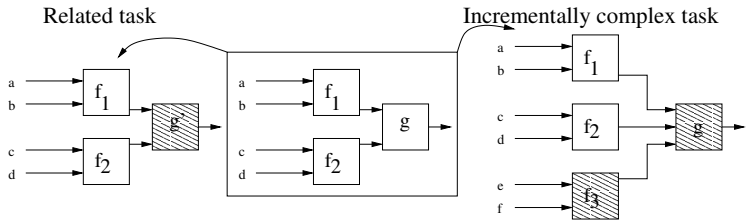


Fig. 1. A modular neural network adapting from $g(f_1, f_2)$ (center) to a related task $g'(f_1, f_2)$ (left) or to a more complex task $g'(f_1, f_2, f_3)$ (right). Shaded modules are to be labelled as "new."

In Fig. 1, modules f_1 and f_2 specialize in corresponding sub-functions during phase-one. To utilise the structural information present in the MNN we must be able to use these two modules in phase-two. However, when we try to learn the next function the modules very quickly lose their specialization[3] and the advantage is lost. When we change the function the corresponding error derivatives are large initially, which results in big changes in parameters of the networks in all the modules, hence we loose the specialization. One of the solutions to this problem is to fix the parameters in the two modules and keep them fixed during the training in phase-two. A less restrictive approach is where we do not fix these parameters but try to control the changes in their magnitude at the beginning of phase-two. If we prevent big changes in these parameters at the beginning, that will give the combination module a chance to adapt to the already specialised modules. Also, in absence of definite information about the relationship between the two functions the second approach is desirable because if they are not related then using the second approach the network can still learn the second function, which might not be possible using the first.

[3] This is similar to the problem of catastrophic forgetting [6] in neural networks which refers to the complete and sudden loss of a network's knowledge, of what it has learnt earlier, in the process of learning a new set of patterns.

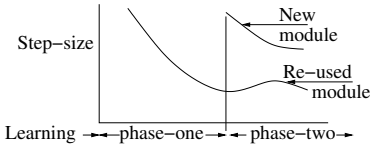


Fig. 2. Typical changes in the absolute values of step-sizes associated with parameters in various modules during adaptation to a related task.

Modified-IRPROP Algorithm: To implement the second approach we need a parameter, associated with each of the weight parameters, controlling the magnitude of changes in these weight parameters. In *Improved Resilient Backpropagation* (IRPROP) [7]) we already have such a parameter called step-size. During training with IRPROP the absolute value of this parameter decreases over time and at later stages of training we have very small values of step-sizes in the network. At the beginning of phase-two of learning the modules that we want to preserve are labelled as "re-used" and others as a "new" modules. We then use step-sizes at the end of phase-one to initialise the step-sizes for the subsequent learning phase for all parameters in "re-used" modules, while "new" modules get their step-sizes initialised normally. This, we argue, is a very natural way of handling learning in modular systems, whereby old modules keep their old characteristics (step-size in this case) while the new modules start with new ones. Using this scheme of initialisation, step-sizes corresponding to the two differently labelled modules exhibit typical behaviours, shown in Fig. 2. In phase-two, step-sizes associated with "new" module start from a high (constant) value, while step-sizes associated with "re-used" module start at very low values, increase first then finally start decreasing again. This indicates some adjustments in "re-used" modules, but not the undesirable catastrophic changes discussed earlier. To illustrate this effect let us now consider an example of each of these tasks.

2.1 Adaptability Towards Related Tasks

To understand the role of modularity in an ANN in such a dynamic environment let us consider an example. For *XOR-OR problem* after learning the *Composite XOR* function $((\overline{a \oplus b}) \oplus (\overline{c \oplus d}))$ in phase-one a network has to adapt to *Composite OR* function $((\overline{a \oplus b}) + (\overline{c \oplus d}))$ in phase-two. Here a, b, c and d are boolean variables and $+$ and $\oplus$ represent OR and XOR functions, respectively. The two boolean functions share common sub-functions and the combination functions OR (g) and XOR (g') are different.

Now for this problem let us compare a fully-connected and a modular structure using the modified-IRPROP, the normal IRPROP and incremental steepest-descent algorithms. In addition we also use cross entropy and mean-squared error functions for these comparisons. Fully-connected structure is an RBF network and the modular network is a combination of three smaller RBF networks, each representing one module in Fig. 1. Number of hidden units are chosen so as to make the number of free parameters the same in both networks. For all these

Table 1. Comparison of adaptability towards related tasks: Cross entropy (CE) or normalised root-mean-squared errors (NRMSE) on training set at the end of phase-two, averaged over 30 runs, for the two structures trained using different learning algorithms. Bold entries in a column represent the significantly better (paired t-test, significance level $\alpha = 0.05$) result in that column.

Algorithm →	**steepest descent**	**IRPROP**		**modified-IRPROP**	
Error Function →	**NRMSE**	**NRMSE**	**CE**	**NRMSE**	**CE**
Structure ↓					
fully-connected	0.48	0.19	0.06	0.19	0.06
modular	**0.31**	0.24	0.04	**0.07**	**0.02**

experiments, phase-two training starts with the weight parameters learnt after phase-one and both phases consist of 100 epochs. Table 1 lists cross entropy or normalised root-mean-squared errors (depending on the error function used) on training set at the end of phase-two, averaged over 30 runs, for the two structures trained using different learning algorithms. Training set errors are used as there are only 16 possible data points for the problem. Modular structure adapts much better than a fully-connected structure if we use incremental steepest descent learning algorithm. With IRPROP both structures adapt equally well. With modified-IRPROP, however, modular structure is much better because we are able to use the modular specialisations in the second task.

2.2 Adaptability Towards Incrementally Complex Tasks

To compare the adaptability of a fully-connected structure and a modular structure to tasks which incrementally become more and more complex[4], let us consider the following three-stage example. In stage-one, the task is to learn f_1, in stage-two the task is to learn $g(f_1, f_2)$ and finally in stage-three the task is to learn $g(f_1, f_2, f_3)$, where $f_1 = \overline{a \oplus b}$, $f_2 = \overline{c \oplus d}$, $f_3 = \overline{e \oplus f}$ and the combination function g is OR. Again a, b, c, d, e and f are boolean variables and $\oplus$ represents XOR. For stage one and two all possible data points (4 and 16, respectively) are used for training and for stage three 50 out of total 64 are used for training and the rest for testing. For stage one there is no modularity in the problem hence we start with two fully-connected RBF networks. In stage-two one of these networks is grown in a modular way, whereby the new inputs go into a separate module, and the other is grown by simply adding more hidden units in the network. These two structures are again grown in a similar fashion in stage-three. In any of these stages the total number of parameters in the two structures is kept same. Figure 3 shows the learning curves for these two structures with mean-squared error function. For stages one and two training error is plotted while for stage three test error is plotted. From these we can see

[4] This is similar to the idea of lifelong learning [8] in robot control tasks, whereby an agent is expected to reduce the difficulty of learning i-th control task by using already acquired knowledge from other tasks.

that the modular structure is able to use the structural information within and performs better than a fully-connected structure. This difference in performance increases with increase in the number of sub-tasks within the overall task. This is observed irrespective of the error function used for training.

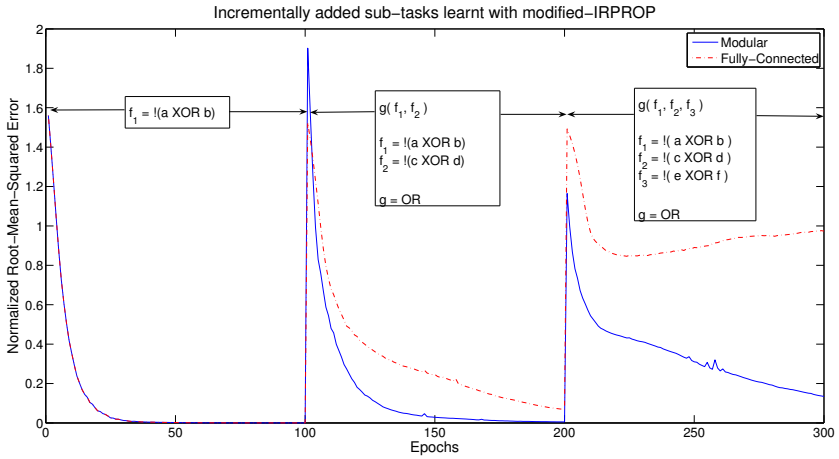


Fig. 3. Average NRMSE of 30 independent runs at different epochs, for the modified-IRPROP learning of modular and IRPROP learning of fully-connected structure.

3 Evolution of Modularity

Here we consider the two dynamic environments again but the second one is made more realistic by making an individual learn incrementally complex task, not in its lifetime but, after every few generations. Previously we observed that making better use of structural information present in a MNN helps it adapt better in these environments. Our aim here is to illustrate how this better adaptability can lead to the evolution of modularity. For this purpose we use an extension of our *co-evolutionary modules and modular neural networks* (CoMMoN) *model* [9, 5]. CoMMoN has a module population or *ModPop* and a MNN population or *SysPop* co-evolving together (Fig. 4). *ModPop* consists of RBF networks and *SysPop* consists of MNNs which are made up of one or more modules from *ModPop*. If an MNN has more than one module, it uses a *Combining-module* (another RBF network) to combine the outputs of these modules. Evolution at *ModPop* level searches for good building blocks and at *SysPop* level it searches for good combinations of these. Within this general co-evolutionary framework, both the structure and parameterisation of MNNs is evolved.

Modules differ from each other in terms of the input connections they have out of all possible inputs for the problem (*AllInps*). Initially they are assigned

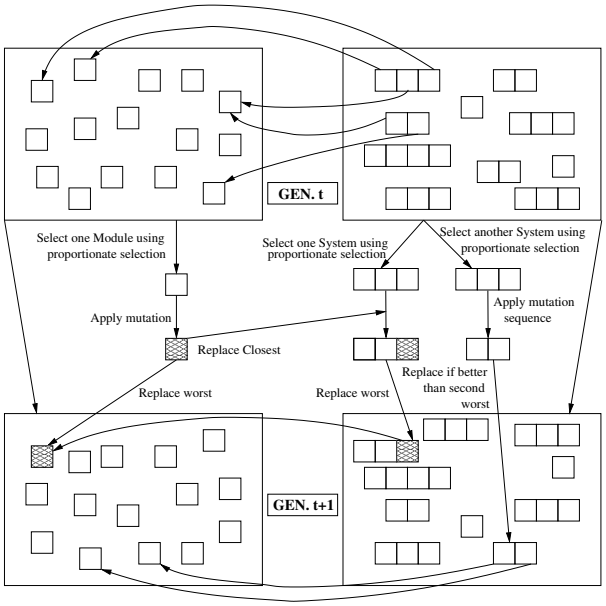


Fig. 4. Steady-state CoMMoN Model : Generation t to generation $t+1$.

these inputs randomly. Centers and widths of hidden units in a module are initialised using K-Means Clustering on the training data points, considering only the inputs which are connected to the module. Weights and bias values are initialised randomly using uniform distribution. In *SysPop*, first each MNN is assigned a random number of modules between 1 and 4 uniformly. If there are more than one modules then a *Combining-module* is added and initialised in a fashion similar to the initialisation of modules in *ModPop*, only the inputs to this module are obtained as the outputs of other modules in the MNN. After initialisation, each MNN is trained using modified-IRPROP and one of the two error functions (cross entropy or mean-squared error) and its fitness is evaluated using the same error function. These fitnesses are then used to evaluate fitness of modules in *ModPop* (Sect. 3.1). In each generation a module from *ModPop* is chosen using proportionate selection. It is then mutated by changing its input positions (each input position is flipped with probability $p_m = 1/AllInps$) and its parameters are reinitialised to obtain *ModChild*. This *ModChild* replaces the worst individual in *ModPop*. A couple of individuals for *SysPop* are then chosen with replacement, using proportionate selection. The first individual is mutated using a mutation sequence (Sect. 3.2) and, if the offspring is fitter than the second worst individual in *SysPop*, it replaces that individual. This mutation is used to make the best use of innovation at the module level. The second system is mutated by swapping the module most similar (based on hamming distance between input connections) to *ModChild* in this individual with *ModChild* and replaces the worst individual in *SysPop*.

3.1 Fitness Assignment

In the first dynamic environment, where individuals (MNNs) need to adapt to a related task, fitness of a MNN is derived from both tasks. We calculate the mean-squared errors (or cross entropy) at the end of each phase of training and sum the inverse of the two values to evaluate an individual. In the second task mean-squared error (or cross entropy) for the current task is used. Fitness for modules in *ModPop* is derived from various MNNs in *SysPop*. We use the sum of fitnesses of best few MNNs (25% of all MNNs in *SysPop*) in which a module participates to evaluate its fitness.

3.2 Mutation Sequence

To encourage simpler structures against more complex structures, the following sequence of mutations (similar to the one used in EPNet [10]) is applied on MNNs in *sysPop*. A particular type of mutation is used only when the preceding type could not produce an offspring which, after partial training, was better than the second worst individual in *SysPop*. The steps involved, in that order, are (a) deletion : a module from the MNN or a node from *Combining-module* is deleted with equal probability, (b) swap: a module in the MNN is swapped with another module from *ModPop*, (c) addition: either a randomly chosen module from *ModPop* is added to the MNN or a new node is added to the *Combining-module*, with equal probability.

4 Simulation Results and Discussion

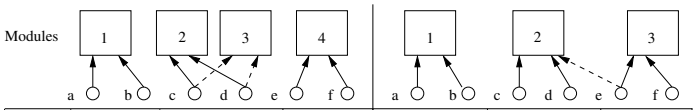
To observe the evolution of modularity we use a structural modularity index or SMI. It indicates how far a given structure is from the modular solution. If the current task has three sub-tasks, for each module in the network this index is defined as: $\text{SMI}_{mod} = \frac{1}{N}(n_1 - n_2 - n_3)$, where n_1, n_2 and n_3 are the number of inputs corresponding to the three sub-tasks, $n_1 \geq n_2 \geq n_3$ and N is the total number of inputs in the problem. If there are only two sub-tasks then n_3 is always zero. The SMI of the MNN is calculated by summing m_{mod}s corresponding to all sub-tasks. If a structure has more than one module for a sub-task then only one is taken into consideration while calculating SMI and this module is the one that (structurally) matches the corresponding sub-task best. Figure 5 illustrates SMI calculations for two MNNs. A structure that matches the problem topology has an SMI value of 1.0 and a fully-connected structure has an SMI value of 0.0.

Various parameters used in experiments for both environments are listed in Table 2. In the first instance where we use adaptability towards related tasks (example from Sect. 2.1) as the fitness measure the best MNN in *SysPop* in all 20 runs (10 each using cross entropy and mean-squared error) at the end of 1000 generations always have an SMI value of 1.0. Although, four runs produce MNNs with an extra module.

To test the adaptability towards incrementally complex tasks the co-evolution -ary model is given 1000, 2000 and 3000 generations to adapt to a task in the

Table 2. Parameter values used for experimentation.

Partial training per generation	20 epochs
ModPop size	120
SysPop size	30
Maximum modules per MNN	4
Maximum hidden units in *Combining-network*	8
Hidden units in modules	5



Sub–task	!(a XOR b)	!(c XOR d)	!(e XOR f)	!(a XOR b)	!(c XOR d)	!(e XOR f)
Module	1	2 or 3	4	1	2	3
SMI_{mod}	1 / 3	1 / 3	1 / 3	1 / 3	1 / 6	1 / 3
SMI	1			5 / 6		

Fig. 5. Errors in connections between modules and inputs in two of the solutions obtained using adaptability towards incrementally complex problem (stage-three) alongside corresponding SMI values. Structure on the left has an extra module and the one on the right has an extra connection.

three stages (Sect. 2.2), respectively. In each stage the collective inputs (all six) are provided and only the target values are changed in between stages. Hence in stage-one (between generation 0 and 999) the task requires feature selection, in stage-two onwards both feature selection and decomposition is required. We use cross entropy in 10 runs and mean-squared error in the other 10 runs as error functions for training and fitness values for evolution. Irrespective of the error function used we obtain modular solutions at the end of both second and third stages. Although the resulting solutions do not match the problem topology exactly. Average SMI values for the best solutions at the end of stage-three for cross entropy and mean-squared error runs are 0.83 and 0.88. Also, t-test ($\alpha = 0.05$) on the two sets of values reveals that the two are not significantly different from each other. A couple of deviations from the exact problem-topology matching structure observed in these results are shown in Fig. 5. Both sets of results indicate that selective advantage based on individuals' ability to adapt in these two environments can results in evolution of modularity. This is observed using both mean-squared error and cross entropy error functions, which indicates the limited influence of the choice of error functions used.

5 Conclusions

Examples of dynamic environments are presented in which the structural information built in a MNN can be better exploited. This is done using the modified-IRPROP algorithm and it results in better adaptability of MNNs than fully-connected structures, independent of various factors involved in training. Error

function and training algorithm (which have been shown to be crucial in learning a static task) are both shown to have limited influence in these environments. Also, with the help of a co-evolutionary model we have shown that in these dynamic environments a selective advantage based on these improved adaptabilities can result in the evolution of modularity, irrespective of these factors.

Previous studies have not been able to show a clear cut advantage of having modularity in neural networks. We argue that the advantage of modularity in neural networks is much more visible in dynamic environments like the ones considered here. Even with such simple dynamic tasks we have been able to show how adaptability benefits from modularity in neural networks. Another, even broader, implication of this work is the partial explanation for the abundance of modularity in natural complex systems. It is argued that dynamic environments like these might be partly responsible for such an abundance.

References

1. Hrycej, T.: Structure of the Brain. In: Modular Learning in Neural Networks. A Modularized Appproach to Neural Network Classification. Wiley, New York (1992) 59–82
2. Wagner, G.P., Mezey, J., Calabretta, R.: Natural Selection and the Origin of Modules. In Callebaut, W., Rasskin-Gutman, D., eds.: Modularity. Understanding the Development and Evolution of Natural Complex Systems. The MIT Press: Cambridge, MA (2005) 33–49
3. Ferdinando, A.D., Calabretta, R., Parisi, D.: Evolving Modular Architectures for Neural Networks. In French, R.M., Sougné, J.P., eds.: Proceedings of the Sixth Neural Computation and Psychology Workshop, Liege, Belgium, Springer Verlag (2001) 253–264
4. Bullinaria, J.A.: To Modularize or Not To Modularize? In Bullinaria, J., ed.: Proceedings of the 2002 U.K. Workshop on Computational Intelligence (UKCI-02), Birmingham (2002) 3–10
5. Khare, V.R., Yao, X., Sendhoff, B.: Multi-network Evolutionary Systems and Automatic Decomposition of Complex Problems. International Journal of General systems (2006) to appear in the special issue on 'Analysis and Control of Complex Systems'.
6. French, R.M.: Catastrophic Interference in Connectionist Networks: Can It Be Predicted, Can It Be Prevented? In Cowan, J.D., Tesauro, G., Alspector, J., eds.: Advances in Neural Information Processing Systems. Volume 6., Morgan Kaufmann (1994) 1176–1177
7. Igel, C., Hüsken, M.: Empirical Evaluation of the Improved Rprop Learning Algorithm. Neurocomputing **50**(C) (2003) 105–123
8. Thrun, S.B., Mitchell, T.M.: Lifelong Robot Learning. Robotics and Autonomous Systems **15** (1995) 25–46
9. Khare, V.R., Yao, X., Sendhoff, B., Jin, Y., Wersing, H.: Co-evolutionary Modular Neural Networks for Automatic Problem Decomposition. In: The 2005 IEEE Congress on Evolutionary Computation, CEC 2005, Edinburgh, Scotland, UK, IEEE Press (2005) 2691–2698
10. Yao, X., Liu, Y.: A New Evolutionary System for Evolving Artificial Neural Networks. IEEE Transactions on Neural Networks **8**(3) (1997) 694–713