Advanced Evolutionary Design of Generalized Recurrent Neural Networks

Simon Vavpotič, Andrej Dobnikar
Andrej.Dobnikar@fri.uni-lj.si
Faculty of Computer and Information Science
University of Ljubljana
Slovenia

Abstract

A new evolutionary algorithm for evolving generalized recurrent neural networks was developed. It has many advanced features, such as forking, exchanging mutation probability distributions and learning, which enable it to find optimal neural network topologies and weights for given problems. We also defined a new parameter, neural network processing speed, which enables us to use networks with one layer of neurons instead of those with many layers. It was proved that the new evolutionary algorithm always finds an optimal solution in a finite number of generations. The proposed algorithm was tested on different problem domains and the results obtained are very promising.

1 Introduction

Evolutionary algorithms are the most popular of non-gradient search methods and they are often used in a simultaneous search for neural network topology and weights. Simple evolutionary algorithms have many difficulties in finding optimal solutions and the population often diverges to local optima. The majority of existing algorithms for evolutionary design of neural networks rely on different types of mutations and different codings of neural networks to chromosomes. We developed a new Advanced Evolutionary Algorithm (AEA) that differs from other evolutionary algorithms with regard to the following built-in features: forking, automatic exchanging of mutation probability distributions, learning and automatic determination of optimal neural network processing speed. The advantages of our new evolutionary algorithm AEA were examined with three groups of experiments: identification of adapted Tomita automata, identification of finite automata with temporal exclusive or functions (TXOR), and robot (ant) control problems. The results obtained were compared with the results published by other authors.

The second chapter outlines the basic structure of an artificial neural network with arbitrary connections and describes a general approaches to neural network design using evolutionary algorithms. The next chapter gives an overview of some advanced features that were used in the construction of our new evolutionary algorithm. In the fourth chapter we describe the convergence features of the AEA and in the last chapter we give an overview of our experimental work. We conclude with some comments and ideas for our future work.

2 Evolving neural networks

2.1. Neural networks

Artificial neural networks are composed of artificial neurons that are based on mathematical models of natural neurons [Haykin, 1998]. An artificial neuron is a nonlinear element with some weighted input connections, an output connection and a transfer function f. If $y_i(u)$ depicts an output value of i-th neuron in the neural network at time u, and $v_{ij}(u)$ depict the neuron's input values and the weight values are given by $w_{ij}(u)$ then the next output value of the neuron is:

$$y_i(u+1) = f\left(\sum_{j=1}^{N_i} w_{ij} \cdot v_{ij}(u)\right), \qquad (1)$$

where N_i gives the number of inputs to the i-th neuron. The bias of the i-th neuron is determined by the weight w_{iNi}, and the input value to this weight is preset to one. The transfer function is a sigmoidal function, for example:

$$f(x) = 1 / (1 + e^{-x}) \qquad (2)$$

Neurons are randomly interconnected. A neural network layer is composed of neurons that process information at

the same time. Processing of layers is ordered so that the first layer processes the information first and the last layer processes the information last.

Neural networks can also be divided according to the direction of signal propagation and neuron interconnections. A feedforward neural networks processes information from inputs to the outputs only. A recurrent neural network can also processes information in the opposite direction. This enables a recurrent neural network to approximate temporal dependencies between input and output samples.

2.2. Evolutionary approach to neural network construction

The basic outline of an evolutionary algorithm is the following [Dobnikar, 1995; Bäck, 2000]. Population $P(t)$ consists of S individuals (solutions). Each individual is a realization of its chromosome (heredity material). In each time step (generation), all individuals in the population are evaluated and a new population is assembled based on genetic operators. The procedure is repeated until a stop condition is met.

A chromosome is an element of a solution space $\boldsymbol{R}$. The evaluation function tests each individual in the population and estimates its performance according to given criteria. Solutions are divided to optimal, sub-optimal and non-optimal. The optimal solutions are those that fully satisfy given criteria. Sub-optimal solutions partially satisfy most of the given criteria and non-optimal solutions do not satisfy most of the criteria.

3 Advanced Evolutionary Algorithm (AEA) features

Evolutionary algorithms are stochastic search methods based on knowledge about natural evolution. There are many ongoing researches that try to improve their efficiency in terms of evaluations needed to find an optimal or sub-optimal solution. In the rest of the chapter we shall introduce some of the basic concepts, that increase their efficiency in terms of speed and accuracy.

3.1 Forking

The idea of forking was first perused in 1993 by Tsutsui and Fujimoto [Tsutsui & Fujimoto, 1993]. Forking enables division of the search space to multiple subspaces. Independent evolutionary processes then investigate the subspaces to find a neural network with suitable topology and weights. Each solution subspace consists of neural networks with arbitrary weight values and equal number of neurons. The solution subspaces are investigated systematically from the subspace with the smallest neural networks to the subspace with the biggest neural networks. Only a subset of solution spaces is searched instantaneously. A special evolutionary strategy determines when a new solution subspace replaces an old one, which is then discarded. The search is stopped when a neural network is found in one of the solution subspaces that solves the desired problem and has the smallest number of neurons. An evolutionary algorithm with forking runs two or more evolutionary processes. First a coarse grain evolutionary process is started over a global solution space R. It has a population $P(t)$ of S individuals. When the coarse grain evolutionary process finds a local solution space R_i around an optimal or a suboptimal solution, this solution space is excluded from the global solution space R. At the same time all individuals from population $P(t)$ that belong to the local solution space R_i are transferred to the local population $P_i(t)$. Some new randomly generated individuals are added to the populations $P(t)$ and $P_i(t)$ so that each of them contains S individuals. The coarse grain evolutionary process is continued over the reduced global solution space and a new independent fine grain evolutionary process is started over the local solution space R_i. This local evolutionary process runs independently until it finds an optimal or a sub-optimal solution in the solution space R_i or the coarse grain evolutionary process terminates it. The number of concurrent local evolutionary processes is limited by the processing capabilities of the computer.

3.2 Adjusting parameters of mutation probability distribution

The evolutionary algorithm AEA has a built-in evolutionary strategy for exchanging probability distributions of mutation. Different studies show that a chosen probability distribution of mutation substantially influences the convergence of the evolutionary process to the optimal solution [Rudolph, 1997]. The current evolutionary algorithms are based on evolutionary strategies that change parameters of mutation probability distribution. It is proven that convergence can be assured to an optimal solution only if parameter changes are very small. Therefore, we avoid changing the parameters of mutation probability distribution. Instead, we use a set of predetermined mutation probability distributions. We developed a new evolutionary strategy that exchanges probability distribution of mutation during the evolutionary process. It assures that the best probability distribution is used during each stage of a neural network evolution. The mutation probability distribution set that we used in our

experiments has the following probability distribution densities:

$$\begin{aligned}
&\xi_1(x, \omega) = E(x, 0, \omega) \\
&\xi_2(x, \omega) = \Lambda(x, 0, \omega, 1, -8, 0, 10, E) \\
&\xi_3(x, \kappa, \omega) = N(x, \kappa, \omega, 1) \\
&\xi_4(x, \omega) = \Lambda(x, 0, \omega, 1, -8, 0, 10, N) \\
&\xi_5(x, \kappa, \omega) = \Lambda(x, \kappa, \omega, 1, -8, 0, 10, E) \\
&\xi_6(x, \kappa, \omega) = N(x, \kappa, 1, 1) \\
&\xi_7(x) = H(x, -8, 0, 10) \\
&\xi_8(x) = \delta(x) \\
&\xi_9(x, \kappa, \omega) = E(x, \kappa, \omega)
\end{aligned} \tag{3}$$

We used the continuous uniform distribution from an interval $[-\omega + \kappa, \omega + \kappa]$ of length 2ω and the middle of κ with the densities:

$$E(x, \kappa, \omega) = \begin{cases} \frac{1}{2\cdot\omega} & ; \quad -\omega + \kappa \le x < \omega + \kappa \\ 0 & ; \quad other \end{cases} \tag{4}$$

and:

$$N(x,\kappa,\omega,\sigma) = \begin{cases} e^{-(x-\kappa)^2/(2\sigma^2)} \cdot \left[\int_{-\omega+\kappa}^{\omega+\kappa} e^{-(x-\kappa)^2/2\sigma^2} dx \right]^{-1} & ; \quad -\omega+\kappa \le x < \omega+\kappa \\ 0 & ; \quad other \end{cases} \tag{5}$$

where the parameter σ is standard deviation. The discrete distribution was defined over the following set of values: $H_x = \{B^{-a}, B^{-a+1}, \ldots, 1, \ldots, B^b\}$. It has three parameters. The parameter a defines the smallest possible exponent, parameter b defines the highest possible exponent and parameter B is the basis. The distribution density $H(x,a,b,B)$ is defined on the basis of the uniform distribution, so that probability of choice of any of the values in the set H_x is the same.

We treat deletion of neural network connection as a special probability distribution $\delta(x)$ that with probability 1 sets a value of a gene to 0. We also defined a hybrid probability distribution with density Λ that is based on a discrete distribution with the following set of values:

$$\{B^{-a} Z, B^{-a+1} Z, \ldots, Z, \ldots, B^b Z\}, \tag{6}$$

where Z is a continuous random variable distributed with a continuous probability distribution with density $\psi(x, \kappa, \omega \cdot B^i, \sigma)$. The probability distribution ψ has the middle κ, the length of interval ωB^i and standard deviation σ. We used uniform and Gaussian probability distribution densities for ψ.

The cyclic sequential exchange of probability distributions starts with one of the probability distribution densities from the set $\Xi = \{\xi_1, \ldots, \xi_9\}$. If the best individual in the population remains the same for a certain predefined number of generations, the density ξ_i is exchanged with the density ξ_{i+1}. AEA uses automatic exchange of probability distribution densities in all the evolutionary processes.

AEA has a special mutation operator that combines properties of the normal mutation [Bäck, 2000] and the differential mutation [Corne et al, 1999]. The differential mutation does not rely on the random generator and a probability distribution. Instead it combines genes in chromosomes of three individuals to produce a chromosome for a new individual. The differential mutation can significantly speedup the evolution for some problems, and is applied with probability of 0.25. When the normal mutation is used the crossover operator is applied to a pair of parents with probability of 0.75 to get two offspring. The normal mutation has a built-in algorithm for exchanging mutation probability distributions. The ordered set of probability distributions is final. The distributions are used in a sequence, so that the first distribution is used as long as it provides convergence. Then it is exchanged with the next distribution from the set in a cyclic manner.

3.3 Learning and evolution

Learning can speedup the evolution. An evolutionary algorithm is used to find a solution in the neighborhood of an optimal solution, then a gradient-based learning method is applied to find the optimal solution. Gradient-based learning algorithms are only used to find exact weight values, but there is no gradient-based method that could determine a neural network topology. Evolutionary algorithm AEA has a built-in learning procedure. Real Time Recurrent Learning (RTRL) algorithm [Gabrijel & Dobnikar, 2003] was integrated in AEA according to the Lamarckian principle of evolution [Bäck et al, 2000]. We chose the Lamarckian principle over the Darwinian principle because we used AEA to design neural networks for static environments. AEA tries to improve the weight values of the best neural networks in the population $P(t)$ by applying a certain predetermined number of learning steps to each of them. If a trained neural network performs better than an untrained one, then the trained neural network replaces the latter.

3.4 Processing speed

Neural network processing speed determines how many times a neural network repeatedly processes the same input sample before processing the next input sample. The present studies neglect the importance of the processing speed, but we show that a single layer neural network with processing speed 1 cannot emulate certain (sequential) logic functions. Multilayer neural networks are used to solve such problems in present studies. Evolutionary algorithm AEA is looking for a solution within a single layer network and uses forking to determine the optimal processing speed.

4 Convergence properties of AEA

The global convergence of evolutionary algorithms was theoretically analyzed with Markov chains in many studies, but Rudolph [Rudolph, 1997] proved that an evolutionary algorithm always finds an optimal solution in a finite number of steps if it satisfies the next four conditions: 1) An arbitrary individual in a population can be selected as a parent for a new population. 2) It must be possible to mutate any solution (this also applies to individuals in a population) in the solution space to any other solution in the solution space by applying a finite number of mutations. 3) The probability of selecting any individual in the current population for the new population must be higher than 0 (zero). 4) The best individual in the current population is always included in the new population. The first, second and third condition can be replaced by a modified second condition: 2*) The mutation operator must be able to mutate each solution in the solution space to any other solution in the solution space in just one mutation. The conditions hold for the evolutionary algorithms that perform searches in finite solution spaces and have discrete time steps [Rudolph, 1997].

The global convergence of evolutionary algorithm AEA is assured if at least one of the probability distributions satisfies the second modified condition. First we have to show that the behavior of the evolutionary algorithm AEA in a solution subspace R_{ij} is the same as the behavior of evolutionary algorithms that operate over finite solution spaces with discrete time steps. The evolutionary algorithm AEA starts an independent evolutionary process EPR_{ij} over each simultaneously searched solution subspace R_{ij}. The solution subspaces are final, because the mutation operator selects values from a finite real number interval $[-\omega, \omega]$. Each finite real number interval is represented as a finite integer number interval in a digital computer. Each solution subspace R_{ij} is a hypercube $[-\omega, \omega]^{i+I}$, where i is the number of neurons in the neural networks in the solution subspace and I is the number of external inputs to the neural networks. Therefore, the length of chromosomes in each population $P_{ij}(t)$ is equal to $i(i+I)$ genes and every evolutionary process EPR_{ij} that satisfy the four conditions can find an optimal solution in its solution space R_{ij} in finite number of steps. Each individual in a population $P_{ij}(t)$ can join to a new population $P_{ij}(t + 1)$ unchanged, since AEA does not apply crossover to 25% of parents and the probability that the mutation does not change an individual is greater than 0. This satisfies the first condition and assures that the solution space is finite.

The second condition is satisfied with the uniform probability distribution ξ_1, which enables the mutation to reach an arbitrary solution in the solution subspace R_{ij} from an arbitrary solution in the same solution subspace. The exchange of probability distributions does not influence the second condition validity, because the probability of each individual in population $P_{ij}(t)$ being preserved in the new population $P_{ij}(t + \Delta)$ is greater than 0, regardless of the current mutation probability distribution ξ_i. The rotation of probability distributions ξ_i is cyclic, therefore the probability distribution ξ_1 is reused in a certain number of generations if the convergence halts. The described mutation exchange procedure is repeated S/4-times in each evolutionary step, where S is the size of population P_{ij}. The reason for repeating the procedure S/4-times is that we want half of the individuals in the population to be parents of a new population, each producing one offspring. New and preserved individuals are then selected to the new population $P_{ij}(t + 1)$. The selection is based on the uniform probability distribution. The probability of an arbitrary individual from the population $P_{ij}(t)$ being preserved in the population $P_{ij}(t + 1)$ is greater than 0. This satisfies the third condition.

The new individuals compete with their parents. The better performing of the two is then included in the new population $P_{ij}(t + 1)$. This satisfies the fourth condition.

The second part of the proof is based on the operation of forking. The neural networks with more neurons have more free parameters than the neural networks with less neurons. Non-existing connections are treated as connections with a weight value of 0. The solution space of neural networks with the greater number of neurons is larger than the solution space of the neural networks with less neurons. Therefore, the evolution of the neural network with more neurons lasts longer and the evolutionary algorithm AEA will always find a neural network with optimal or close to the optimal number of neurons in forward search. When a neural network with a suboptimal number of neurons is found,

AEA will continue to search backwards and will look for the solutions with the lower number of neurons. Therefore, an optimal solution would always be found. This phenomenon was experimentally examined and proved. Figure 1 illustrates it on a problem of automaton identification. The results are given in the number of evaluations that equals the number of generations multiplied by the size of population.

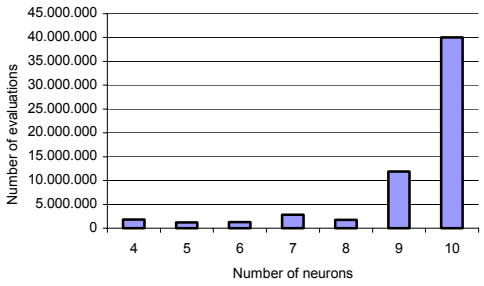


Figure 1: Comparison of the number of the necessary evaluations to find optimal weight values for the neural networks with different numbers of neurons that identify Tomita automaton 6. (see details in Chapter 5)

AEA is more complex compared to the other evolutionary algorithms that simultaneously search for optimal topology and optimal weight values of neural networks. Nevertheless, it is for the simple tasks on average approximately as fast as simple evolutionary algorithms, and is able to solve more complex tasks. It is also much faster in solving complex problems, because it uses forking, exchanges mutation probability distributions, learning and also automatically determines the necessary neural network processing speed.

5 Experiments

The evolutionary algorithm AEA was tested on two problem domains: finite automata identification and robot (ant) control. The results were compared with the results obtained by evolutionary algorithms GNARL [Angeline et al, 1994] and GA2DR [Pujol, 1999] and the results obtained by a gradient-based algorithm GARNN [Gabrijel & Dobnikar, 2003].

The evolutionary algorithm AEA was tested from two points of view. First, we compared the statistically evaluated results to the results obtained by other algorithms that solved the same problems. All statistical evaluations are based on 10 independent evolutionary runs. They contain two measurements: the average neural network size and the average convergence speed. It is important to note that the neural networks obtained by AEA had the same size for a given problem in any of the evolutionary runs, because AEA was always able to find an optimal solution. Second, we measured the partial speedups of the advanced features of the AEA.

The evolutionary algorithm AEA used forking in all of the experiments, because it is its key feature that enable it to determine the optimal neural network size and the optimal processing speed. The variable z indicating the number of neurons was set to the initial value of 2 before each evolutionary search. The number of simultaneously searched solution subspaces L was set to 10. The set of possible processing speeds was limited to 1 and 2 to shorten the duration of the evolution. The population size in all experiments was 100 individuals. All the neural networks used in the automata identification problems had one input and one output. The input values were 0 and 1, but the output values were from the real valued interval between 0 and 1. The output values were converted to the discrete values of 0 and 1. The values lower then 0.5 were discretized to 0 and the rest were converted to 1.

5.1 Finite automata identification

The identification experiments were performed on four Tomita automata (4', 5, 6, 7'), and four temporal XOR functions (with d = 0,1,2,3), where the desired value at time u is the XOR function of the inputs at times $u - d$ and $u - d - 1$. There are seven Tomita automata alltogether that are used as acceptors for automata languages. Five of them are not strongly connected. Therefore, some internal states cannot be reached from an arbitrary internal state. The automata 4 and 7 were altered by [Gabrijel & Dobnikar, 2003] into 4' and 7' to allow online identification (Figure 2). We also used AEA to identify temporal XOR functions (d = 0,1,2,3). The Ξ set (Eq. 3) of probability distributions was used in all of the experiments. The weight values for neural networks were chosen from the real valued interval [-200, 200]. The evaluation function was based on the error function:

$$Err(t) = | o_d(t) - o(t) | , \qquad (7)$$

where $o_d(t)$ was desired output value and $o(t)$ was the discretized neural network output. The errors were summed over the test sample sequence of input and output value pairs. The best performing neural network in the population had the lowest total error. The test sequence had 125,000 samples. The first 1000 samples were used in evolution and the remaining 124,000 samples were used for testing. An evolved neural

network was considered as a sub-optimal solution if its total error on the testing sequence was 0. The additional condition for an optimal neural network was the lowest possible number of neurons.

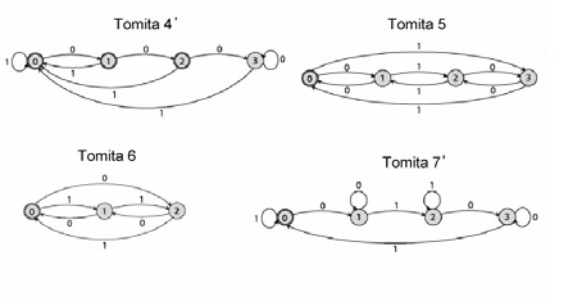


Figure 2: The Tomita automata state transition diagrams. The double circled states have the output letter 1 and the other states have the output letter 0. The altered Tomita automata are marked with '.

Four series of experiments were performed. The first series compared the sizes of the resulting neural networks obtained by AEA, GNARL and GARNN on the problems of Tomita automata 4', 5, 6, and 7' and logical functions TXOR-0 through TXOR-3 . Figure 3 shows the comparison of the different neural network sizes. The best results were obtained by AEA.

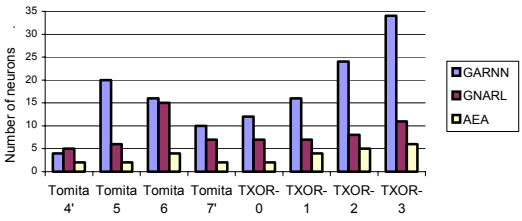


Figure 3: Comparison of the neural network sizes. The evolutionary algorithm AEA clearly shows the best results.

The next series compared the convergence speed of AEA and GNARL. GARNN was not included in the comparison because it is a gradient-based method and it is much faster than evolutionary algorithms. Figure 4 compares the evolutionary algorithm AEA to the evolutionary algorithm GNARL. The convergence speed was measured for Tomita automata identification problems 4', 5, 6, and 7'.

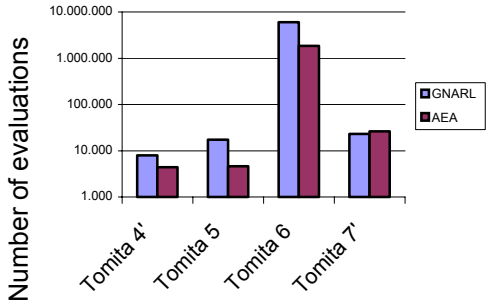
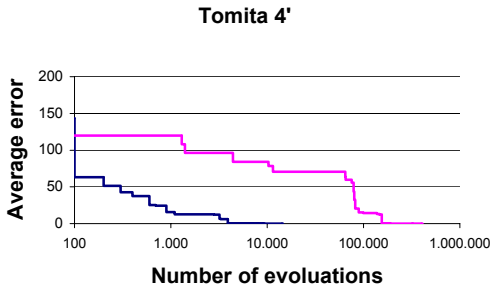
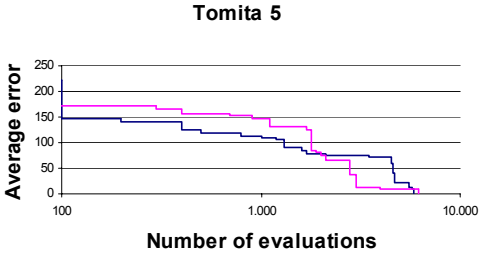


Figure 4: The comparison of the number of necessary evaluations to identify Tomita automata 4', 5, 6, and 7'.

The experiments in the last two series were designed to measure the speedup of the evolution due to use of learning and due to the exchanging of mutation probability distributions. While learning was switched on, the RTRL learning algorithm was used to train 5% of the best performing individuals in the population. The average progress of evolution with learning over ten evolutionary runs was compared to the average progress of evolution without learning over ten evolutionary runs. The influence of learning was first measured for the identification problems of Tomita automata 4', 5, 6, and 7'. The results are given in Figure 5.





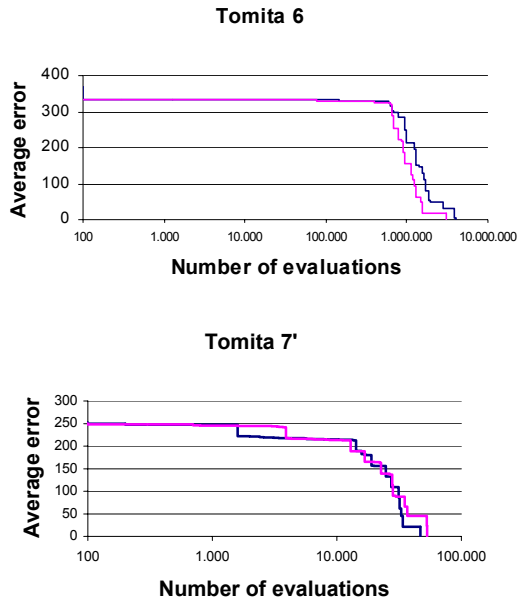
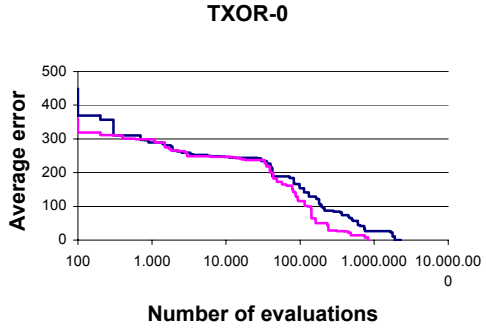
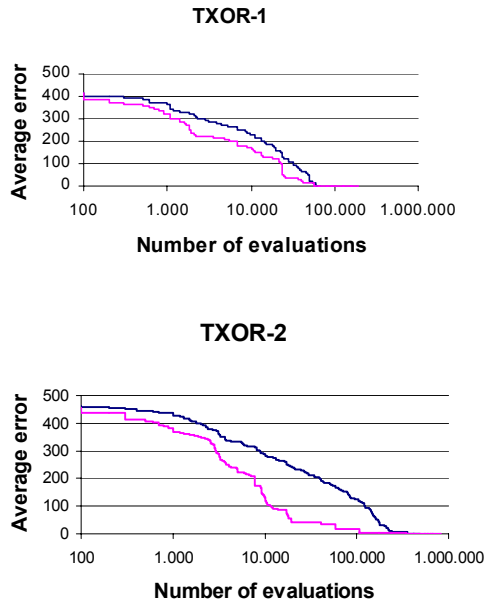


Figure 5: Average progress of evolution of optimal neural networks that solve Tomita automata 4', 5, 6, and 7' identification problems. The dotted curves show average evolution with learning and the continuous curves without learning.

In all the experiments except for the Tomita automaton 4' identification learning helped evolution. Learning speeded up the evolution up to 30%, especially for the more difficult problems such as Tomita automaton 6 identification.

Next, the influence of learning was measured for the identification problems of temporal XOR functions with delays from 0 to 3.





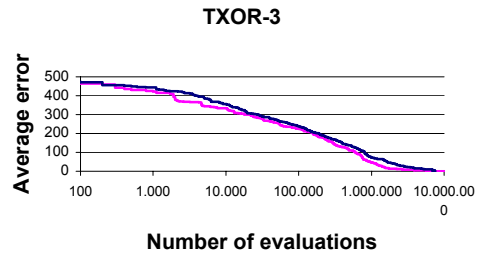


Figure 6: Average progress of evolution of optimal neural networks that solve TXOR-0 through TXOR-3 identification problems. The dotted curves show average evolution with learning and the continuous curves without learning.

The results are shown on Figure 6. The last set of experiments evaluated the performance of evolutionary algorithm AEA with and without exchanging probability distributions. A hundred evolutionary runs were performed for each of the Tomita automata 4', 5, 6, and 7' and for each of the logical functions from TXOR-0 through TXOR-3. The first part of the test was designed to see how the evolutionary algorithm AEA performs with fixed probability distributions. We made

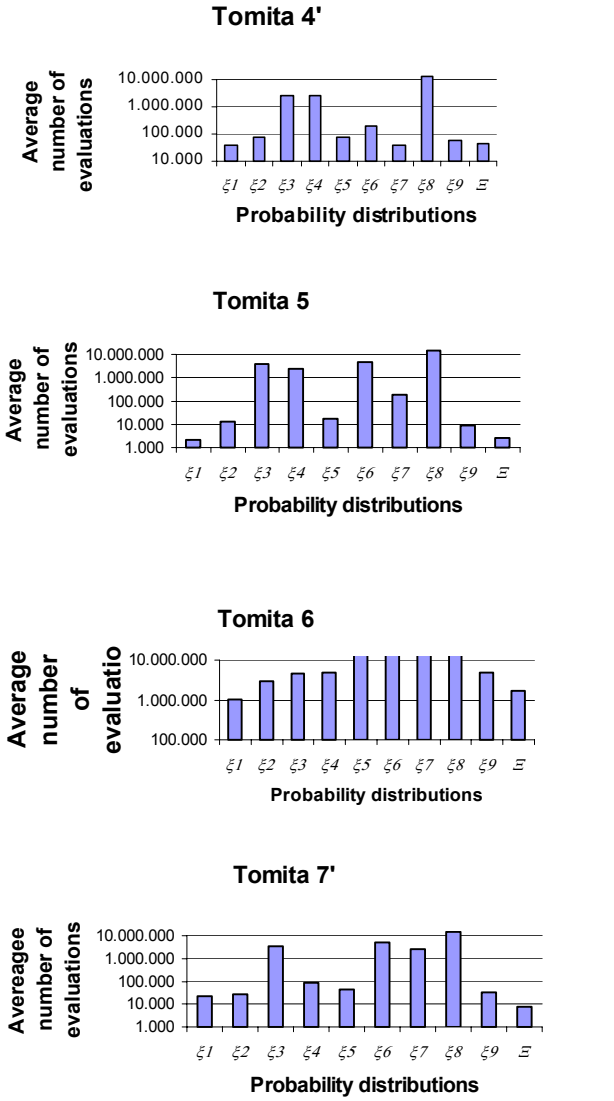


Figure 7: Average evolution durations for fixed probability distributions and for the Ξ set of probability distributions for Tomita automata 4', 5, 6, and 7' identification problems.

10 evolutionary runs for each probability distribution from the Ξ set. In the second part of the experiment, we made ten evolutionary runs with exchanging probability distributions. The results are shown in Figures 7 and 8.

The results of the last set of experiments show that using the best single probability distribution provides on average, slightly faster evolution than the set of probability distributions, but for an unknown problem the exchanging of the mutation probability is the most suitable, because no single probability distribution is the best for all problems.

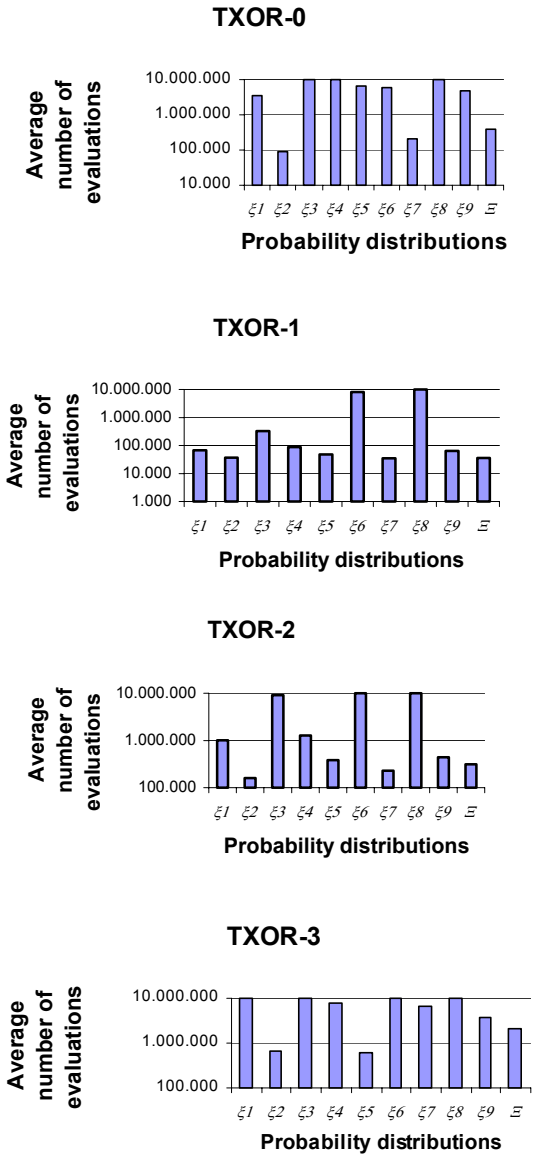


Figure 8: Average evolution durations for fixed probability distributions and for the Ξ set of probability distributions for TXOR-0 through TXOR-3 identification problems.

5.2 Ant problem

One of the most popular robot control problems is the ant problem [Angeline et al, 1994]. We used the evolutionary algorithm AEA with all its advanced features turned on to solve this problem. An ant moved in a dicretized 2D space (Figure 9), which contained a food trace. The size of the ant and the size of a food particle is one field. Each field in the 2D space is assigned a value of 1 if it contains food and a value 0 if it does not contain food. If the ant moves into a field

with food, it eats the food and resets the value of the field to 0. The ant is allowed 200 time steps to eat the whole food trace of 89 particles. It has four control actions: move, turn 90 degrees left, turn 90 degrees right and stand still. One penalty point is collected for each action, except if it moves to a field with food. The neural network obtained with AEA is shown on figure 10. There are four control outputs. The one with the highest value is always selected.

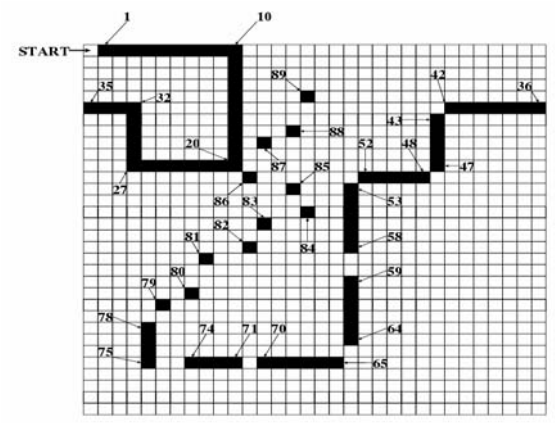


Figure 9: 2D space with a food trace for the ant problem

The "standard" Ξ set of probability distributions is used with AEA. The weight values for neural networks are chosen from the real valued interval [-500, 500]. AEA evolved an optimal neural network controller with 4 neurons on average in 5,033,130 evaluations. The average progress of ten AEA evolutionary runs is shown in the Figure 11.

The fastest of the ten evolutionary runs lasted only 2,474,800 evaluations. The controller had four neurons and "ate" all the food particles in 198 time steps. On the other hand, the best GNARL neural network controller had 12 neurons and needed 319 time steps to eat all the food [Angeline et al, 1994]. Therefore, GNARL had only managed to find a partial solution. We also compared the AEA result to the result obtained by GA2DR [Pujol, 1999]. GA2DR needed 4,373,600 evaluations to find an 8-neuron neural network controller that collected all the food particles in 200 time steps.

6 Conclusion

The evolutionary algorithm AEA found optimal solutions for all of the given problems. The obtained

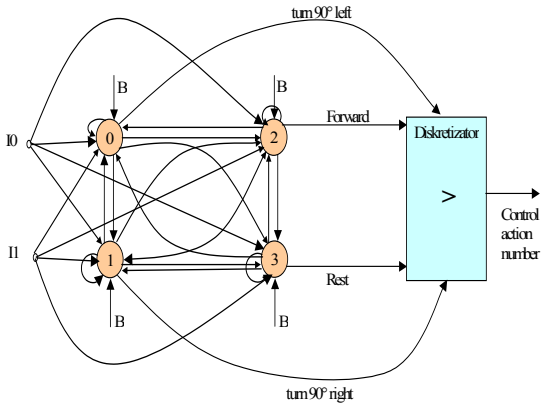


Figure 10: A neural network controller with 4 neurons (outputs), evolved by AEA.

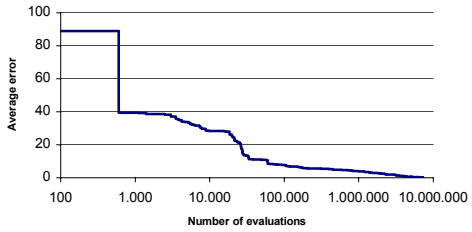


Figure 11: Average progress of the evolution over 10 evolutionary runs

neural networks had at most half as many neurons as the compared solutions [Angeline et al, 1994, Pujol, 1999, Gabriel & Dobnikar, 2003]. The AEA solutions have also better generalization capabilities, because they solve equally difficult problems with many less neurons, which have memory capabilities. Therefore, smaller neural networks must rely more on concepts and associations. A simulation of smaller neural networks in a digital computer also requires much less processor time. Its complexity enables the evolutionary algorithm AEA to solve hard problems faster then the compared evolutionary algorithms. On the other hand, AEA is not significantly slower on simple problems.

The evolutionary algorithm AEA develops recurrent neural networks with no limitations regarding the topology. It is suitable for searching neural network solutions of various complex problems. The only current limitation of the evolutionary algorithm AEA is the evolution of a single layer neural network. But this limitation is shared with most of the other evolutionary algorithms that evolve neural networks. On the other hand, single layer neural networks can solve the hardest

problems, if they are able to operate at suitable processing speeds.

The further development of our algorithm will be focused on upgrading the method for exchanging the mutation probability distributions based on past statistical evidences. We are also developing a model of a generalized multi-layer neural network and a method for optimal distribution of neurons to specific neural network layers.

References

[Angeline et al, 1994] P. J. Angeline, G. M. Sauders, J. B. Pollack: *An evolutionary algorithm that constructs recurrent neural networks*. IEEE Trans. on Neural Networks, 5, (1):54-65, 1994.

[Bäck et al, 2000] T. Bäck, D. B. Fogel & Z. Mihalewicz: *Evolutionary Computation*. Institute of Physics Publishing, Bristol & Philadelphia, 2000.

[Corne et al, 1999] D. Corne, M. Dorigo and F. Glover: *New Ideas in Optimization*. McGraw-Hill, 1999.

[Dobnikar, 1995] A. Dobnikar: *Evolutionary design of application-specific neural networks: A genetic approach*. Neural Network World, 5(1):41-50, 1995.

[Gabrijel & Dobnikar, 2003] I. Gabrijel, A. Dobnikar: *On-line Identification and Reconstruction of Finite Automata with Generalized Recurrent Neural Networks*, Neural Networks, vol. 16, No. 1, 101-121, Jan., 2003.

[Haykin, 1998] S. Haykin: *Neural networks: A Comprehensive Foundation,* Prentice Hall, 1998.

[Pujol, 1999] J. C. F. Pujol: *Evolution of artificial neural networks using a two-dimensional representation*. Doktorat, School of Computer Science, University of Birmingham, 1999.

[Rudolph, 1997] G. Rudolph: *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovac, 1997.

[Tsutsui & Fujimoto, 1993] S. Tsutsui, Y. Fujimoto: *Forking genetic algorithm with blocking and shrinking modes.* Proceedings 5, International Conference on Genetic algorithms, 206-213, 1993.

[Yao, 1999]X. Yao: Evolving Neural Networks, Proceedings. of the IEEE, 1999.