Evolving Neural Networks for Online Reinforcement Learning

Jan Hendrik Metzen[1], Mark Edgington[2], Yohannes Kassahun[2], and Frank Kirchner[1,2]

[1] Robotics Lab, German Research Center for Artificial Intelligence (DFKI)
Robert-Hooke-Str. 5, D-28359, Bremen, Germany
[2] Robotics Group, University of Bremen
Robert-Hooke-Str. 5, D-28359, Bremen, Germany

Abstract. For many complex Reinforcement Learning problems with large and continuous state spaces, neuroevolution (the evolution of artificial neural networks) has achieved promising results. This is especially true when there is noise in sensor and/or actuator signals. These results have mainly been obtained in offline learning settings, where the training and evaluation phase of the system are separated. In contrast, in online Reinforcement Learning tasks where the actual performance of the systems during its learning phase matters, the results of neuroevolution are significantly impaired by its purely exploratory nature, meaning that it does not use (i. e. exploit) its knowledge of the performance of single individuals in order to improve its performance during learning. In this paper we describe modifications which significantly improve the online performance of the neuroevolutionary method Evolutionary Acquisition of Neural Topologies (EANT) and discuss the results obtained on two benchmark problems.

1 Introduction

Reinforcement Learning (RL) is concerned with deciding which action a (virtual or real) agent should take in a given state of an environment in order to maximize its long-term reward. The strategy an agent follows is called its *policy*. Traditionally, methods from the domain of Temporal Difference (TD) Learning [10] have been most popular for solving RL problems. TD learning is essentially a search in value function space where the policy is *indirectly* optimized by changes in estimated value of a state-action pair. In contrast, methods which search directly in the space of policies have gained more attention recently. Examples of these kind of methods are policy gradient [11] and neuroevolution [15]. In *neuroevolution* (NE), the policy an agent follows is represented as an artificial neural network (ANN), which represents a mapping from state to action. NE optimizes an ANN (and thus the policy) by applying an Evolutionary Algorithm (EA), which modifies either the weights or both the topology and the weights of the ANN. Typically, the policy represented by an ANN is not updated after each step as usually done in TD learning but after each episode or after a fixed number of steps. This is due to the fact that EAs usually assess the performance of an individual (in this case the ANN) as a whole using a fitness function. In episodic, stochastic RL problems, it is most natural to use an approximation of the expected reward per episode as fitness function, e. g. by following the given policy for a certain number of episodes and approximate the expected long-term reward by the average of the actually obtained reward per episode. A critical question is for how many episodes a policy is followed before its expected reward per episode is estimated [6].

It has been shown that NE can outperform TD methods in domains with large and/or continuous state and action spaces especially when sensors and/or actuators are subject to noise [2,12]. However, the comparison between TD learning and NE has usually been done only for offline RL. *Offline RL* means that the agent has a training phase where its actual performance does not matter and only after this phase is it confronted with the real problem where it should act optimally. In contrast, in *online RL* the agent has to act as well as it can from the very beginning, meaning that it has to maximize the obtained reward starting from the first step. Online RL is important since not all tasks can be shaped into an offline RL problem. For instance, the dynamics of real world tasks like robot control are often not completely known or too complex to be accurately simulated, and thus an agent would likely need to learn online in the real world.

Online RL is more challenging for NE mainly due to the following reason: At each moment in time the EA used in NE operates on a whole set of individuals (ANNs), the population. This population normally contains a broad range of ANNs and usually only a few of them represent policies that acquire a large long-term reward. In offline RL, at the end of the training phase only the best individual of the entire training is chosen to perform in the testing phase (the phase in which its performance matters), and because of this a large long-term reward in the testing phase is obtained. However, this is not possible in online RL since it is not known beforehand which ANNs represent a good policy, and it is therefore necessary to test bad individuals frequently in order to assess their fitness. This decreases the overall performance of any NE method in online RL drastically.

In this paper, we will discuss how an NE method can be modified such that the online performance of the method is greatly improved. We will first give a review of works which have applied NE in the context of online RL (Section 2), explain the basic EANT algorithm and its extension to online RL (Section 3), and present the results obtained by offline and online EANT as well as by the TD method Sarsa(λ) on two benchmarks (Section 4). We will conclude with a brief outlook (Section 5).

2 Review of Works

In this section, we discuss works in which neuroevolutionary methods have been applied in the context of online RL and the closely related RL with real-time demands. For a general review of the works in the evolution of neural networks we refer to Yao [15].

Whiteson et al. [14] conducted an empirical study on how to balance the trade-off between exploration and exploitation in the context of neuroevolution. In order to transfer mechanisms for balancing exploitation and exploration from TD learning to neuroevolution, the level at which those mechanisms are applied is modified: instead of applying these mechanisms on the level of individual actions, they were applied on the level of episodes, in which entire policies are assessed holistically. This was necessary because evolutionary methods have no notion of the value of individual actions but only of whole policies (i. e. they perform a search in policy space and not in value function space). Three different mechanisms were compared (ϵ-greedy selection, softmax selection, and interval estimation) on two benchmark problems (mountain car [10] and server job scheduling [13]) using the neuroevolutionary method NEAT [7]. It was found that all three mechanisms clearly outperform the offline version of NEAT (where all policies get the same number of evaluations) in terms of maximizing the overall accumulated reward. Furthermore, softmax selection and interval estimation performed roughly the same, but both performed significantly better than ϵ-greedy selection.

Realtime demands in the context of RL are addressed by Stanley et al. [8]. They describe how NEAT can be modified in order to meet the realtime demands of a multi-agent continuous-state machine learning game called NeuroEvolving Robotic Operatives (NERO). The main modification of the NEAT algorithm is that instead of replacing an entire population at once, only single individuals are replaced. The worst performing agent among those members of the population that have been evaluated sufficiently is replaced by a new agent created by NEAT's standard mechanism for producing offspring from a given population. The idea of replacing only one individual at once originates from the area of evolution strategies [1] and is known as *steady-state* evolution.

The main contribution of this work is to show that balancing exploitation and exploration and steady-state evolution are two orthogonal concepts and can be meaningfully combined. Furthermore, a new mechanism for balancing exploitation and exploration that is especially suited for neuroevolution is proposed.

3 Evolutionary Acquisition of Neural Topologies (EANT)

3.1 Offline-EANT

In this section, we discuss the EANT algorithm[1] [3]. In the context of this work, we refer to the algorithm proposed in [3] as *Offline-EANT*.

The basic procedure of Offline-EANT is outlined in Algorithm 1. Its structure is similar to most generational EAs: first, the initial population of *num_indiv* individuals is created. Then, for each generation, the respective population is evaluated for a given (fixed) number of episodes *num_evals*. Based on these evaluations, the fitness of each individual is estimated. In `Create-Next-Generation`,

[1] The source code for EANT is available under the URL *http://sourceforge.net/projects/mmlf/*

```
Offline-EANT(num_evals, num_indiv):
    $population \leftarrow$ Create-Individuals($num\_indiv$)
    while $True$ do
        for $i \in \{0, \ldots, num\_evals\}$ do
            $individual \leftarrow$ Choose-Individual($population$)
            $fitnessSample \leftarrow$ Evaluate-Individual($individual$)
            Update-Estimated-Fitness($individual$,$fitnessSample$)
        $population \leftarrow$ Create-Next-Generation($population$)
```

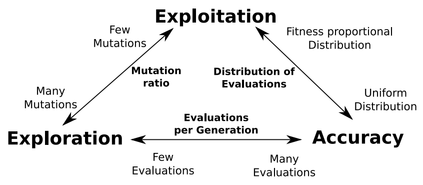
Algorithm 1. The offline (generational) EANT algorithm

some of the members of the population are selected with respect to their fitness and these individuals and their offspring (generated using the genetic operators) form the next population. Creation of the initial population and of the next population based on the current one are described in more detail in Metzen et al. [5]. The individuals EANT acts on are usually encoded in a genome and evaluating them involves developing these genotypes into the corresponding phenotypes. In principle, EANT can be combined with a magnitude of genetic encodings of neural networks. However, in practice EANT has typically been combined with the Common Genetic Encoding (CGE) [4].

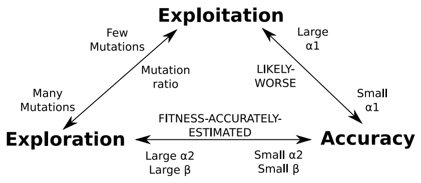
The most important properties of Offline-EANT are (1) that the evolution is divided into two time scales, a smaller one on which the ANN's weights are optimized and a larger one on which the topology of the ANN is optimized [3], and (2) that (optionally) the population is divided into species and the concept of fitness sharing is applied to these species as proposed by Stanley [7].

3.2 Tradeoffs in Offline-EANT

Each evaluation of an individual (policy) in Offline-EANT consists of playing a full episode with this policy, and results in one sample of the fitness function. In deterministic environments with fixed start state, the same individual will always get the same fitness, and thus there is no need to evaluate an individual more than once. In stochastic environments, however, the fitness samples are "noisy" and one evaluation per individual is not sufficient for estimating an individual's fitness accurately. Thus, two questions arise: What is an optimal value for *num_evals* (the evaluations per population) and how should the evaluations be distributed among the individuals? In neuroevolution, it turns out that there is more than just a *exploration-exploitation* trade-off. There is also the issue of estimating an individuals fitness accurately (see Figure 1a). Choosing the value of *num_evals* influences the *exploration-accuracy* trade-off since setting *num_evals* to large values will increase the accuracy of fitness estimates while decreasing the extent to which new network structures and weights are explored, and vice versa. On the other hand, how evaluations are distributed among individuals influences the *exploitation-accuracy* trade-off, since



(a)



(b)

Fig. 1. The three conflicting objectives of neuroevolution (exploitation, exploration, and accuracy), and how they can be controlled by Offline-EANT (a) and Online-EANT (b)

distributing them equally among the individuals will maximize average fitness estimation accuracy but will not exploit information accrued during evaluation, while other distributions will necessarily decrease the estimated fitness accuracy for some individuals. The trade-off between exploitation and exploration is influenced both indirectly via the accuracy and directly via the mutation ratio since less mutations will increase the number of evaluations of individuals which have been found before to perform well (i. e. exploitation) but will decrease the occurrence of new individuals (i. e. exploration), and vice versa. Whiteson et al. [14] (see Section 2) have mainly addressed the exploitation-accuracy trade-off by comparing different ways of distributing the evaluations among the individuals (like softmax selection and interval estimation). These approaches can easily be realized by implementing `Choose-Individual` in Algorithm 1 accordingly. The exploration-accuracy trade-off, on the other hand, could be addressed by manually adjusting *num_evals*. However a systematic, automatic approach would be highly desirable.

3.3 Online-EANT

In this section, we discuss how the EANT algorithm can be modified in a way that allows it to perform more efficiently in online RL tasks. The resulting algorithm, called *Online-EANT*, is summarized in Algorithm 2 and has the form of a *steady-state evolution*, i. e. the evolution is no longer divided into generations. Instead, there is a continuous change in the population, some new individuals being "born", some older individuals "dying" (as in nature). The population is divided into two disjoint sets of individuals. The first set is the set of "mature" individuals. The fitness of these individuals has been accurately estimated, and they are able to create offspring. The other set is the set of "adolescent" individuals. These are the individuals which are currently tested by the algorithm in the environment. None of the adolescent individuals is able to create offspring, and none of the mature individuals will ever be evaluated in the environment again[2].

[2] At this point, it is assumed that the fitness function is stationary, i. e. that the fitness distribution of an individual does not change over time.

```
Online-EANT(num_indiv):
    $adolescents \leftarrow$ Create-Individuals($num\_indiv$)
    $matures \leftarrow \emptyset$
    while $True$ do
        $individual \leftarrow$ Choose-Randomly($adolescents$)
        $fitnessSample \leftarrow$ Evaluate-Individual($individual$)
        Update-Estimated-Fitness($individual$,$fitnessSample$)
        if Likely-Worse($individual$,$matures$) then
            $adolescents.replace$($individual$,Create-New-Offspring($matures$))
        else if Fitness-Accurately-Estimated($individual$) then
            if Fit-Enough($individual$,$matures$) then
                $matures.add$($individual$)
                if To-Many-Individuals($matures$) then
                    $matures.remove\text{-}oldest$()
            $adolescents.replace$($individual$,Create-New-Offspring($matures$))
```

Algorithm 2. The online (steady-state) EANT algorithm

Initially, there are only adolescent individuals (the quantity is determined by the parameter *num_indiv*). For each of these adolescent individuals, there are two possibilities: either they become mature after some time (i. e. evaluations) or they "die" prematurely. Which of these options occurs is determined based on the set of fitness samples obtained by the individual in a way similar to Stagge [6]. For this purpose, two criteria are checked:

(1) If the hypothesis h_1 that the individuals true fitness is below the average estimated fitness of the mature individuals is valid with a certain significance level α_1, the individual is considered to have "died" prematurely and is replaced by a new individual which is created based on the set of mature individuals using EANT's standard procedure of selection and applying genetic operators. The hypotheses h_1 is checked by the function `Likely-Worse`.

(2) If hypothesis h_2 that the individuals current fitness estimate differs from its true fitness by less than a factor β is valid with a certain significance level α_2, it is checked whether the individual's estimated mean fitness is larger than the average fitness of the mature individuals. If that is the case, the individual is added to the mature individuals. If afterwards, the number of mature individuals exceeds a certain number, the oldest (*not* the least fit) mature individual is removed from *matures*. If the individual's estimated mean fitness is not larger than the average fitness of the mature individuals, it is not becoming a mature individual but is replaced by a new individual. The hypotheses h_2 is checked by the function `Fitness-Accurately-Estimated`.

3.4 Handling of Tradeoffs in Online-EANT

As in Offline-EANT (see Section 3.2), there are three conflicting objectives in Online-EANT: exploration, exploitation, and accuracy. In Online-EANT, the exploitation-accuracy and the exploration-accuracy trade-offs are explicitly addressed: (1) The method `Likely-Worse` ensures that no evaluations are "wasted" on individuals which have turned out to be not competitive (on the basis of the obtained fitness samples of the individuals). An individual is considered to be not competitive if the hypothesis h_1 is valid with a certain significance level α_1 (the p-value is computed using an one-tailed, one-sample t-test). The smaller α_1, the more cautious the algorithm is in discarding an individual (i. e. the fitness of an individual needs to be estimated more accurately before it is discarded) but at the expense of exploiting the knowledge of good individuals less. Thus, α_1 controls the exploitation-accuracy trade-off. (2) The method `Fitness-Accurately-Estimated` controls how accurately the fitness of a (competitive) individual is estimated. The fitness is considered to be estimated accurately if the hypothesis h_2 is valid with a certain significance level α_2 (the p-value is computed using a two-tailed, one-sample t-test). The smaller the required significance level α_2 and the allowed error rate β, the more accurately the fitness of an individual is estimated, but at the cost of a reduced amount of exploration. Thus, α_2 and β control the exploration-accuracy trade-off.

The exploitation-exploration trade-off is addressed explicitly in Online-EANT via the mutation rate, but it can be influenced also implicitly via the two other trade-offs: decreasing α_1 and increasing α_2 at the same time will result in roughly the same accuracy but will increase exploitation at the cost of exploration. The relationship between the three issues is depicted in Figure 1b. Whether Online-EANT's way of handling the trade-offs yields better online RL performance than formerly studied approaches is discussed in Section 4.

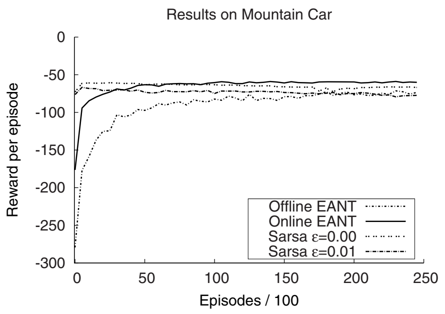
4 Results

Mountain Car. We have tested three different RL methods in the Mountain Car benchmark [10]: The TD method Sarsa(λ), Offline-EANT, and Online-EANT. For Sarsa, we used the Cerebellar Model Articulation Controller (CMAC) [10] function approximator with a superposition of 10 independent tilings, replacing eligibility traces with decay rate $\lambda = 0.95$, a learning rate of $\alpha = 0.5$, and a discount factor of $\gamma = 1.0$. Two different values for ϵ (the ratio of choosing a random, non-greedy action) have been tested, namely $\epsilon = 0.0$ and $\epsilon = 0.01$. The initial Q-values were all set optimistically to 0 to enforce initial exploration. For Offline-EANT, we used a population size of 20, evaluated each individual for 10 episodes, and disabled fitness sharing. For Online-EANT, we created 20 individuals initially, set the required significance levels α_1 and α_2 to 0.05 and the allowed error rate β to 20%. The maximum number of steps an individual was allowed to take to reach the goal was restricted to 500. If the goal was not reached after this number of steps, the next individual took over control of the car at the current position. Thus, no artificial ending of an episode was necessary and neuroevolution could not get stuck in an unfeasible policy which never reached the goal.

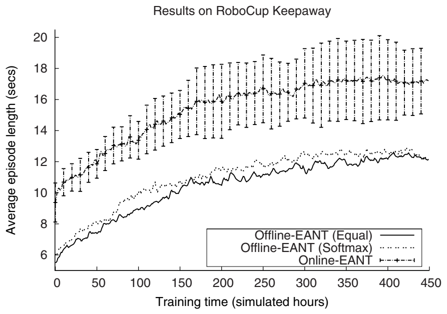
Figure 2a shows the reward per episode of the three methods in the benchmark (the plotted values are the averages over 50 independent runs for each method and are smoothed using a moving window average in order to reduce fluctuations induced by the stochastic start state of the benchmark). Online-EANT obtains significantly more reward per episode than Offline-EANT over the whole 25000 episodes of evaluation ($p < 7 * 10^{-5}$). This shows that Online-EANT's choice of evaluating more promising individuals more often does not only improve initial performance but does also not interfere with the long-term progress of the evolution. Compared to Sarsa ($\epsilon = 0.0$), Online-EANT obtains less reward per episode in the early phase of a trial (the difference is significant during the first 4000 episodes ($p < 0.003$)) but achieves better performance in the long run (the difference is significant after 10000 episodes ($p < 0.05$)). In contrast, Offline-EANT's performance remains significantly worse than Sarsa's ($p < 0.05$) over the whole evaluation time. Sarsa's superior performance for in the initial phase of the trials can be explained by the fact that it makes use of the instantaneous reward supplied by the environment (instead of assessing the policies as a whole) while its worse performance in the long run might be explained by the fact that it is more easily trapped in locally optimal policies (in particular if the learning rate ϵ is set to 0). However, setting ϵ to 0.01 did not yield in an improved online performance (see Figure 2a). Compared to the results presented in [14], Online EANT achieves better online performance than the combination of NEAT with any policy selection strategy. However, since Offline-EANT achieves better performance than the performance reported for Offline-NEAT in [14], this gives no indication on whether the proposed online modifications for EANT are better than different policy selection strategies. Because of this, we analyze this issue in the next section on a more selective problem.

RoboCup Keepaway. Keepaway is part of the RoboCup Soccer Simulator and was introduced as a benchmark by Peter Stone et al. [9]. Metzen et al. [5] have shown that the combination of Offline-EANT and CGE can outperform the results which have been published for the temporal difference learning method Sarsa(λ) and for the neuroevolutionary method NEAT [12] in the Keepaway benchmark problem when given enough training and applied in an offline scenario. In contrast to this offline assumption, we compare in this section the *online* performance of Online-EANT with two versions of traditional, generational EANT (Offline-EANT). For Online-EANT, we created 50 individuals initially and set the maximal number of mature individuals to 25. We set α_1 and α_2 to 0.05 and β to 10%. For Offline-EANT, we used a population size of 50. We compared two different strategies for balancing exploitation and estimation accuracy in Offline-EANT, namely giving all individuals the same number of evaluations and applying softmax policy selection like proposed by Whiteson [14]. The number of episodes per generation was set to $num\_evals = 2000$, giving each individual 40 evaluations in the average. For both Online- and Offline-EANT, we enabled fitness sharing.

Figure 2b shows the performance of the three methods (the plotted values are the averages over 6 independent runs for each method and are smoothed using a moving window average in order to reduce fluctuations induced by the stochastic start state of the benchmark). Online-EANT's performance is



(a)



(b)

Fig. 2. Performance Comparison on the Mountain Car (left) and RoboCup Keepaway (right) benchmark. Each curve is an average over 50 (Mountain Car) and 6 (Keepaway) independent runs.

significantly better than Offline-EANT's ($p < 0.009$) and Offline-Softmax-EANT's performance ($p < 0.05$) during the whole $450h$ of evaluation. In contrast, Offline-Softmax-EANT's performance is only in the first 6 hour significantly better than Offline-EANT's ($p < 0.05$). Altogether, the results show that proposed method, Online-EANT, improves the online performance of EANT significantly while simply using Offline-EANT with a modified policy selection strategy like Softmax policy selection does not (or only to a very small amount which could not be detected in 6 runs).

5 Conclusions and Outlook

In this paper, a method of extending the neuroevolutionary method EANT has been proposed that permits efficient learning in online RL tasks. In terms of maximizing the accumulated reward, this extension outperforms previous approaches in which softmax policy selection is applied to classical offline neuroevolution. The discussed method mainly addresses the issue of maximizing the accumulated reward during learning. However, for applying RL algorithms in general and neuroevolution in particular to online RL, further issues need to be addressed: when learning in the real world, some actions might be dangerous to the system in certain situations. For instance, when learning to control a helicopter, it must be assured that no policy will explore actions that lead to a crash of the system (which would have virtually infinite costs). In this case, some kind of domain knowledge (an approximate model, for instance) needs to be incorporated into the method. Furthermore, it would be interesting to investigate how the proposed method performs in environments with different characteristics, e. g. environments with a non-stationary fitness function.

References

1. Beyer, H.-G., Schwefel, H.-P.: Evolution strategies - a comprehensive introduction. Natural Computing: an International Journal 1(1), 3–52 (2002)
2. Gomez, F.J., Schmidhuber, J., Miikkulainen, R.: Efficient non-linear control through neuroevolution. In: Proceedings of the 17th European Conference on Machine Learning (ECML), Berlin, Germany, September 2006, pp. 654–662 (2006)
3. Kassahun, Y.: Towards a Unified Approach to Learning and Adaptation. PhD thesis, Institute of Computer Science and Applied Mathematics, Christian-Albrechts University, Kiel, Germany (February 2006)
4. Kassahun, Y., Edgington, M., Metzen, J.H., Sommer, G., Kirchner, F.: A common genetic encoding for both direct and indirect encodings of networks. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO 2007), pp. 1029–1036 (2007)
5. Metzen, J.H., Edgington, M., Kassahun, Y., Kirchner, F.: Analysis of an evolutionary reinforcement learning method in a multiagent domain. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008), Estoril, Portugal, pp. 291–298 (May 2008)
6. Stagge, P.: Averaging efficiently in the presence of noise. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 188–200. Springer, Heidelberg (1998)
7. Stanley, K.O.: Efficient Evolution of Neural Networks through Complexification. PhD thesis, Artificial Intelligence Laboratory. The University of Texas at Austin., Austin, USA (August 2004)
8. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the nero video game. IEEE Trans. Evolutionary Computation 9(6), 653–668 (2005)
9. Stone, P., Kuhlmann, G., Taylor, M.E., Liu, Y.: Keepaway soccer: From machine learning testbed to benchmark. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 93–105. Springer, Heidelberg (2006)
10. Sutton, R., Barto, A.: Reinforcement Learning. An Introduction. MIT Press, Massachusetts (1998)
11. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems 12, pp. 1057–1063 (1999)
12. Taylor, M.E., Whiteson, S., Stone, P.: Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006), pp. 1321–1328 (2006)
13. Whiteson, S., Stone, P.: Evolutionary function approximation for reinforcement learning. Journal of Machine Learning Research 7, 877–917 (2006)
14. Whiteson, S., Taylor, M.E., Stone, P.: Empirical studies in action selection with reinforcement learning. Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems 15(1), 33–50 (2007)
15. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE 87(9), 1423–1447 (1999)