Analysis of an Evolutionary Reinforcement Learning Method in a Multiagent Domain

Jan Hendrik Metzen
German Research Center for Artificial Intelligence (DFKI)
Robert-Hooke-Str. 5, D-28359 Bremen, Germany
jhm@informatik.uni-bremen.de

Mark Edgington
University of Bremen
Robert-Hooke-Str. 5, D-28359 Bremen, Germany
edgimar@informatik.uni-bremen.de

Yohannes Kassahun
University of Bremen
Robert-Hooke-Str. 5, D-28359 Bremen, Germany
kassahun@informatik.uni-bremen.de

Frank Kirchner
German Research Center for Artificial Intelligence (DFKI)
Robert-Hooke-Str. 5, D-28359 Bremen, Germany
frank.kirchner@informatik.uni-bremen.de

ABSTRACT

Many multiagent problems comprise subtasks which can be considered as reinforcement learning (RL) problems. In addition to classical temporal difference methods, evolutionary algorithms are among the most promising approaches for such RL problems. The relative performance of these approaches in certain subdomains (e. g. multiagent learning) of the general RL problem remains an open question at this time. In addition to theoretical analysis, benchmarks are one of the most important tools for comparing different RL methods in certain problem domains. A recently proposed multiagent RL benchmark problem is the RoboCup Keepaway benchmark. This benchmark is one of the most challenging multiagent learning problems because its state-space is continuous and high dimensional, and both the sensors and the actuators are noisy. In this paper we analyze the performance of the neuroevolutionary approach called Evolutionary Acquisition of Neural Topologies (EANT) in the Keepaway benchmark, and compare the results obtained using EANT with the results of other algorithms tested on the same benchmark.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Connectionism and neural nets*

General Terms

Algorithms

Keywords

Reinforcement Learning, Neuroevolution

Cite as: Analysis of an Evolutionary Reinforcement Learning Method in a Multiagent Domain, Jan Hendrik Metzen, Mark Edgington, Yohannes Kassahun, and Frank Kirchner, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp.291-298.

1. INTRODUCTION

Evolutionary algorithms can be considered as reinforcement learning algorithms, where the fitness value of an individual is an accumulated reward received by the individual after it has operated in a given environment [26]. Unlike in traditional reinforcement learning algorithms, where a reward signal is provided after each action executed by the individual, in evolutionary algorithms a fitness value (return) is assigned to the individual at the end of the life time of the individual or after the individual has carried out a sequence of actions (an episode). This property of evolutionary algorithms make them directly applicable to episodic reinforcement learning tasks such as game playing, where they search for optimal value functions or optimal policies directly in the space of value functions or policies, respectively.

It has been shown using standard benchmark problems that a combination of neural networks and evolutionary methods (neuroevolution) can perform better than traditional reinforcement learning methods in many domains, especially in domains which are non-deterministic and only partially observable [5, 22]. One advantage of neuroevolutionary methods is that the policy is represented using an artificial neural network (ANN), which is useful for learning tasks involving continuous (noisy) state variables. This is due to the fact that ANNs provide a straightforward mapping between states perceived by the sensors and actions executed by the actuators. Additionally, ANNs are robust to noise: since their units are typically based upon a sum of several weighted signals, oscillations in the individual values of these signals do not drastically affect the behavior of the network [12].

In this paper, we present a performance evaluation of the neuroevolutionary method Evolutionary Acquisition of Neural Topologies (EANT) on the Keepaway benchmark [19], which is a subproblem of the RoboCup Soccer Simulator. This benchmark problem is challenging since the states are continuous and only partially observable and the sensors and actuators of the agents are noisy. The paper is organized as follows: first, a review of work in the area of neuroevolution is given. Then, an introduction to EANT is provided along with a brief description of the Keepaway benchmark problem. After this, experimental results on the performance

of EANT on the benchmark problem are presented, and the effects of several components of EANT are analyzed. Finally, some conclusions and a future outlook are provided.

2. REVIEW OF WORK IN NEUROEVOLUTION

The field of Neuroevolution (NE) can be divided into two major areas of research: in the first area, the structure of the ANN is kept fixed and only the weights are optimized by the EA. In the second area, both the structure and the weights are evolved in parallel. This paper will focus on the second, more general area. For a review of the work in the evolution of neural networks covering both areas see Yao [27].

All methods which evolve the structure of the network assume a certain type of *embryogeny*. The term embryogeny refers to the growth process which defines how a genotype maps onto a phenotype. According to Bentley and Kumar [3], three different types of embryogenies have been used in evolutionary systems: external, explicit and implicit. *External* means that the developmental process (i. e. the embryogeny) itself is not subjected to evolution but is hand-designed and defined globally and externally with respect to the genotypes. In *explicit* (evolved) embryogeny the developmental process itself is explicitly specified in the genotypes, and thus it is affected by the evolutionary process. Usually, the embryogeny is represented in the genotype as a tree-like structure following the paradigm of genetic programming. The third kind of embryogeny is *implicit* embryogeny, which comprises neither an external nor an explicit internal specification of the growth process. Instead, the embryogeny "emerges" implicitly from the interaction and activation patterns of the different genes. This kind of embryogeny has the strongest resemblance to the process of natural evolution.

The following encodings utilize an external embryogeny: Angeline et al. developed a system called GNARL (GeNeralized Acquisition of Recurrent Links) which uses only structural mutation of the topology, and parametric mutations of the weights as genetic search operators [1]. The main shortcoming of this method is that genomes may end up having many extraneous disconnected structures that have no contribution to the solution. The Neuroevolution of Augmenting Topologies (NEAT) [18] starts with networks of minimal structures and increases their complexity along the evolution path. The algorithm keeps track of the historical origin of every gene that is introduced through structural mutation. This history is used by a specially designed crossover operator to match genomes which encode different network topologies. Unlike GNARL, NEAT does not use self-adaptation of mutation step-sizes. Instead, each connection weight is perturbed with a fixed probability by adding a floating point number chosen from a uniform distribution of positive and negative values.

As opposed to these encodings with external embryogeny, the following encodings adopt an explicit (internal) embryogeny: Kitano's grammar based encoding of neural networks uses Lindenmayer systems (L-systems) [11] to describe the morphogenesis of linear and branching structures in plants [10]. Sendhoff et al. extended Kitano's grammar encoding with a recursive encoding of modular neural networks [16]. Their system provides a means of initializing the network weights, whereas in Kitano's grammar based encoding, there is no direct way of representing the connection weights of neural networks in the genome. Gruau's Cellular Encoding (CE) method is a language for local graph transformations that controls the division of cells which grow into an artificial neural network [6]. The genetic representations in CE are compact because genes can be reused several times during the development of the network and this saves space in the genome since not every connection and node needs to be explicitly specified in the genome. Defining a crossover operator for CE is still difficult, and it is not easy to analyze how crossover affects the subfunctions in CE because they are not explicitly represented.

The last class of embryogeny (the implicit one) is utilized by the following encodings: Vaario et al. have developed a biologically inspired neural growth based on diffusion field modelling combined with genetic factors for controlling the growth of the network [23]. One weak point of this method is that it cannot generate networks with recurrent connections or networks with connections between neurons on different branches of the resulting tree structure. Nolfi and Parisi have modelled biological development at the chemical level using a reaction-diffusion model [13]. This method utilizes growth to create connectivity without explicitly describing each connection in the phenotype. The complexity of a structure that the genome can represent is limited because every neuron is directly specified in the genome. Other work in implicit embryogeny has borrowed ideas from systems biology, and simulated Genetic Regulatory Networks (GRNs), in which genes produce signals that either activate or inhibit other genes in the genome. Typical works using GRNs include those of Bongard and Pfeifer [4] and Reisinger [14].

3. EVOLUTIONARY ACQUISITION OF NEURAL TOPOLOGIES

The Evolutionary Algorithm we use is *Evolutionary Acquisition of Neural Topologies* (EANT, *http://sourceforge.net/projects/mmlf/*) [7]. EANT uses a unique dual-timescale technique in which the neural network's connection weights are optimized on a small timescale, and the neural network's structure evolves gradually (on a larger timescale). EANT starts with networks of minimal complexity, which are gradually complexified. Although EANT can be used with any arbitrary genetic encoding which can represent neural networks, we have chosen to use the *Common Genetic Encoding* [8], an encoding with features that make it well suited for use in the evolution of neural networks.

3.1 Common Genetic Encoding

The Common Genetic Encoding (CGE) is a general framework for encoding and modifying neural networks (the *phenotypes*). It can be applied as an encoding with external or explicit embryogeny [9]. The encoding has important properties that make it suitable for evolving neural networks [8]: It is *complete* in that it is able to represent all types of valid phenotype networks, and it is *closed*, i. e. every valid genotype represents a valid phenotype. Furthermore, the encoding is *closed under genetic operators* such as structural mutation and crossover (defined below) that act upon the genotype. Another important feature of CGE is that an encoded phenotype can be directly evaluated without needing first to decode the phenotype from the genotype. Details on how this is done are discussed by Kassahun [7]. While we use CGE here to encode networks that are interpreted as neural networks, CGE is not limited to this purpose [9].

CGE encodes a network using a *linear genome*. This genome consists of a string of genes, where each gene is either a *vertex gene* (representing a vertex in the network, also called a *neuron gene*), an *input gene* (representing an input in the network), or a *jumper gene* (representing an explicit connection between vertices). Each gene possesses one or more attributes which provide information about the gene. For example, each gene can possess a weight (which modifies its output), and each vertex gene possesses a unique id

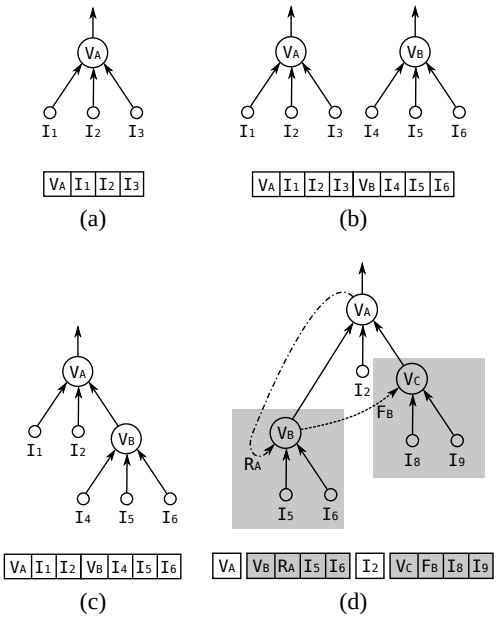

Figure 1: Different examples illustrating CGE: (a) a simple three input network, (b) two subnetworks in parallel, (c) two subnetworks in cascade, (d) two subnetworks connected to a main network, including forward and recurrent jumper connections. Each example includes a neural network representation, and the corresponding CGE genome representation. All vertex genes in the genome representations possess a "number of inputs" attribute $d_{in} = 3$.

and a "number of inputs" attribute d_{in}.

A genome can be subdivided into one or more *subgenomes*, each of which is a valid genome in itself. The simplest sort of subgenome consists of a vertex gene followed by several input genes, one input gene for each input of the vertex represented by the vertex gene (see Figure 1a). Such a subgenome encodes a simple network with one *output vertex* (a vertex gene whose output is an output of the network). To encode a network containing two of these networks in parallel, two identical copies of the subgenome are simply concatenated together to form a new genome that encodes a two-output network (see Figure 1b). On the other hand, to connect one of these simple networks (network B) as an input to another simple network (network A), one of the input vertices in network A's genome is replaced with the entire genome of network B (see Figure 1c).

In the last example, the vertex in network B (v_B) is implicitly connected to the vertex in network A (v_A). The output of v_B can have only one such implicit connection to another vertex. In more complex networks, there is typically a need to define more than one connection from a vertex's output to other vertices. This is accomplished in CGE by the use of jumper genes. A jumper gene possesses a "source ID" attribute which refers to the unique id possessed by a vertex gene. A jumper gene acts like a "virtual copy" of the vertex gene to which it refers, providing the output of this vertex as an input to another vertex. Therefore, if one wants to add an additional connection from v_B's output to a vertex other than v_A (for example, to v_C in Figure 1d), a jumper gene referring to v_B can be put in the place of one of v_C's input genes. Jumper genes are either forward or recurrent. The forward jumper F_B provides the output of v_B immediately to v_C, while the recurrent jumper R_A provides v_A's output to v_C in the next evaluation of the network.

3.2 Genetic Operators

Several genetic operators have been developed for CGE which operate on one or two linear genomes to produce another linear genome [7]. The three operators we used are the *parametric mutation*, the *structural mutation* and the *structural crossover*.

3.2.1 Parametric Mutation

The Parametric Mutation operator performs a random perturbation of the connection weights of each gene. Each genome stores a learning rate σ, which acts as the standard deviation of each weight's modification: $w_i' = w_i + N(0, \sigma)$, where $N(0, \sigma)$ is a real number drawn from a normal distribution with mean 0 and standard deviation σ. The learning rate itself is modified during the parametric mutation according to the rule $\sigma' = \sigma * N(0, 1)$. This kind of parametric mutation allows for the self adaptation of strategy parameters, a paradigm proposed in the field of Evolutionary Strategies [15].

3.2.2 Structural Mutation

The Structural Mutation operator inserts either a new randomly generated subgenome, a forward jumper, or a recurrent jumper after an arbitrarily chosen neuron gene. The inserted subgenomes consist of a new neuron gene (with unused unique id) followed by an arbitrary number of input and jumper genes. This kind of structural mutation differs from the one proposed in NEAT [18] by the fact that whole subnetworks can be introduced at once without the need to add all their nodes and edges separately. This might help the method to find network topologies of sufficient complexity faster (see Section 5.4), though at the cost of missing the simplest topology potentially. Initially, the weights of the newly added structures are set to 0, in order to avoid that these can affect the genome's overall performance. When a forward jumper gene is added to the genome, care is taken to avoid closed cycles of forward jumper genes, since this would cause infinite looping problems later on during the evaluation of the network.

3.2.3 Structural Crossover

The third genetic operator defined is the Structural Crossover operator. This operator exploits the fact that structures which originate from the same ancestor structure have some parts in common. These parts are detected using the unique id of the genes, which act as historical markers. By aligning the common parts of two randomly selected structures, it is possible to generate a third structure that contains the common and disjoint parts of the two mother structures. The resulting structure formed in this way maps to a valid phenotype network. This type of crossover was introduced and is used by Stanley [18].

3.3 Elements of EANT

Our implementation of EANT follows the basic steps of an Evolutionary Algorithm: initialization, selection, mutation, crossover, and fitness evaluation. In this work, we use Stochastic Universal Sampling [2] as selection mechanism. Mutation and crossover is accomplished by the genetic operators presented in Section 3.2. Some parts of EANT are designed specifically for the needs of neuroevolution; these are presented in this section. For a complete overview over EANT we refer to [7].

3.3.1 Initialization

The first thing that must be done in an evolutionary algorithm is to create an initial population P_0 of individuals that contains a sufficient amount of diversity. Since we are following the approach of minimizing dimensionality through incremental growth

from minimal structure [18], the members of P_0 should be as simple as possible. For a task with n state dimensions (inputs) and m possible actions (outputs), we create the initial population as follows: first, we generate a "proto" individual g_p consisting of m neuron nodes (encoding the outputs) and $n * m$ input genes (connecting all inputs with each output neuron). g_p encodes a network with no hidden layers. All individuals of the initial population P_o are descended from this proto individual by applying the structural mutation operator up to five times onto a copy of g_p. This guarantees that we have a sufficiently diverse initial population with fairly simple structures. Furthermore, since any two randomly selected individuals (and all members of later populations) have a common ancestor, they can be aligned by the structural crossover operator.

3.3.2 Exploitation / Exploration

As in classical reinforcement learning methods [21], in neuroevolutionary approaches there is a trade-off between exploitation of existing structures (i. e. optimizing the weights of the encoded networks) and exploration of new structures (i. e. generating new networks). While in conventional neuroevolutionary approaches both occur at the same time, we explicitly divide the evolutionary process into two phases (as proposed by Kassahun [7]): in the *exploitation phase*, the structural mutation and crossover are disabled (hence no new structures are created) and thus, only the parameters (i. e. the weights) of existing structures are modified. Furthermore, population wide selection does not take place. After a certain number of generations $n_{exploit}$, the exploitation phase is finished and an *exploration phase* is started, where the structural operators are activated for $n_{explore}$ generations.

The purpose of this approach is to give newly created structures time to optimize their weights, before they have to compete population-wide with all other structures in the exploration phase. The quantity $n_{explore}$ is usually set to 1 while the choice of $n_{exploit}$ is a trade-off: larger values allow structures a better optimization of their weights before competing, while smaller values increase the frequency at which new, promising structures occur. We chose a value of $n_{exploit} = 5$, which has proven to be a good compromise.

3.3.3 Speciation and Fitness Sharing

As mentioned above, in the exploitation phase there is no population wide competition, since this would penalize newly introduced structures whose weights are unoptimized. Instead, at the beginning of an exploitation phase, the population is divided into so called *species* (an idea introduced to the field of neuroevolution by Stanley [18]). This division is done using a distance measure defined on the level of genotypes:

$$d(g_1, g_2) = 1 - \frac{|N(g_1) \cap N(g_2)| + |J(g_1) \cap J(g_2)|}{|N(g_1) \cup N(g_2)| + |J(g_1) \cup J(g_2)|}$$

where g_1, g_2 are arbitrary genotypes, $N(g)$ is the set of neuron genes, and $J(g)$ is the set of jumper genes contained in g. An individual is part of a species if and only if its mean distance to the members of this species is below a certain threshold. If there is no such species, this individual forms a new species. During the exploitation phase, competition (i. e. selection based on the fitness values) only takes place within a species, i. e. between individuals with similar structures.

Speciation gives new structures time to optimize their weights during the exploitation phase. However, it does not prevent one successful structure/species from taking over the whole population. To prevent this, the selection mechanism in the exploration phase uses *fitness sharing* [18]. Fitness sharing means that an individual with fitness f, which is part of a species P_s and a population P_c, gets assigned a new fitness: $f' = f * (1 - \frac{|P_s|-1}{|P_c|})$, where $|P_s|$ and $|P_c|$ are the sizes of P_s and P_c, respectively. Fitness sharing decreases the fitness of individuals which are members of a large species. This modification is motivated by the observation that in nature, species share an ecological niche and thus, members of the same species must compete for the same resources. If a species gets larger, the selection pressure on its individuals increases, which can be seen as a decrease of their fitness.

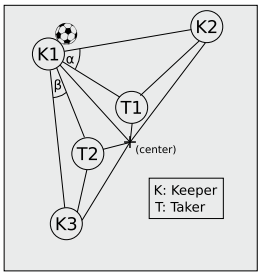

Figure 2: The state variables provided by the benchmark: 11 distances, and two angles.

4. THE KEEPAWAY BENCHMARK

Keepaway is part of the RoboCup Soccer Simulator and was introduced as a benchmark by Peter Stone et al. [19]. In the 3 versus 2 (3vs2) Keepaway benchmark, a team of three *keepers* attempts to maintain possession of a ball in a two-dimensional playing-field (usually $20m \times 20m$) while a team of two *takers* tries to intercept the ball. The actions performed by the keeper that is currently in possession of the ball are controlled by a learned policy π whereas the actions performed by the keepers not in possession of the ball are determined by a fixed policy given by the benchmark, as are the actions performed by the takers. The benchmark is subdivided into episodes, each starting in a similar (but not identical) start state and ended when the ball is either intercepted by a taker or goes out of the bounds of the field. The goal of the learning system is to learn a policy π that optimizes the behavior of the active keeper (i. e. maximizes the episode duration). This episode duration is a machine-independent simulation time and can be used as a fitness/quality measure of the policy implemented by the agent. Since both sensors and actuators are exposed to noise, Keepaway is a highly stochastic benchmark, and the duration of a single episode is not a reliable estimate of a policy's fitness; rather, the average duration of multiple episodes played with the same policy is a better estimate. One question that arises is whether Keepaway can be considered to be a multiagent learning task, since the policy controls only one agent at a time. However, Stone et al. [20] state that the benchmark's complexity comes not so much from the individual learning task, but from the multiagent component. In other words, learning all three keepers in parallel is actually harder than learning one keeper with two pre-trained teammates.

The state provided by the benchmark consists of 13 continuous state variables: 6 of the distances between the players, 5 distances from the players to the center of the field, and two angles associated with the passing lanes of the ball-possessing keeper (i.e. keeper #1). These state variables are depicted in Figure 2. The policy can choose from one of three predefined, discrete macro-actions, which are performed by keeper #1: hold the ball, pass the ball to keeper #2, and pass the ball to keeper #3. Since the "pass" action might last longer than one time step (0.1sec), the problem is a semi-Markov decision process (SMDP). Furthermore, since the sensors are ex-

posed to noise, the problem is a *partially observable SMDP*. All results reported below have been obtained using version 0.6 of the Keepaway benchmark.

Following the method described in Section 3.3.1, networks which can act as policy for Keepaway in EANT have been encoded as follows: In 3vs2 Keepaway, the policy maps a state consisting of 13 variables onto one of 3 actions. If one wants to encode such a policy in an ANN, the ANN should have 3 outputs and 13 inputs. Therefore, the proto individual g_p consists of exactly 42 genes. Given a certain state, the action corresponding to the output with maximal activation is chosen. Thus, the network encodes directly the policy and no explicit value function is involved.

5. RESULTS AND COMPARISON

In this section we present results obtained with EANT in the Keepaway domain and relate them to results of other reinforcement learning method published by other authors. Unfortunately, the results published by other authors are often not directly comparable since they use different versions of Keepaway and differ in some important parameters, e. g. the field size, the number of players per team, and whether the sensors are noisy or not. In order to relate the performance of EANT to these results, we applied EANT in different settings, namely in Partially Observable and Fully Observable Keepaway on a $20m \times 20m$ field, and in Partially Observable Keepaway on a $25m \times 25m$ field.

5.1 Partially Observable 3vs2 Keepaway

Partially observable (PO) 3vs2 Keepaway is the standard setting for the Keepaway benchmark: three keepers play against two takers on $20m \times 20m$ field, and both sensors and actuators are exposed to noise. We performed 8 independent runs of EANT in the standard Keepaway benchmark; the results are shown n Figure 3a. The average performance of a generation's champion (i. e. the best performing individual of a population) over the 8 runs is plotted, as well as the corresponding standard deviation. The average episode duration after $800h$ training time was $14.9sec$ with a standard deviation of $1.25sec$, indicating that the method always converges to reasonable solutions. The mean episode duration after $800h$ training time of the best run was $16.6sec$, while the mean episode duration of the worst run was $12.9sec$.

5.2 Fully Observable 3vs2 Keepaway

Fully observable (FO) 3vs2 Keepaway differs from the standard setting in Section 5.1 in that the sensors are free of noise. The actuators, however, are still subject to noise and the start state is stochastic. It is important to note that (even though this setting is usually referred to as fully observable Keepaway) the state is still not truly Markovian since it does not include player velocities. However, Taylor et al. [22] argue that the state is "effectively Markovian" since players have low inertia and the field has a high coefficient of friction which means that velocity does not help agents learn in practice.

We performed 9 independent runs of EANT in FO 3vs2 Keepaway benchmark; the results are shown in Figure 3b. Plotted is the performance of the champion of a generation (i. e. the best performing individual of a population), averaged over the 9 runs, as well as the corresponding standard deviation. Unsurprisingly, removing the sensor noise simplifies the task for the learning system. The average episode duration after $800h$ training time was $19.2sec$ with a standard deviation of $1.16sec$.

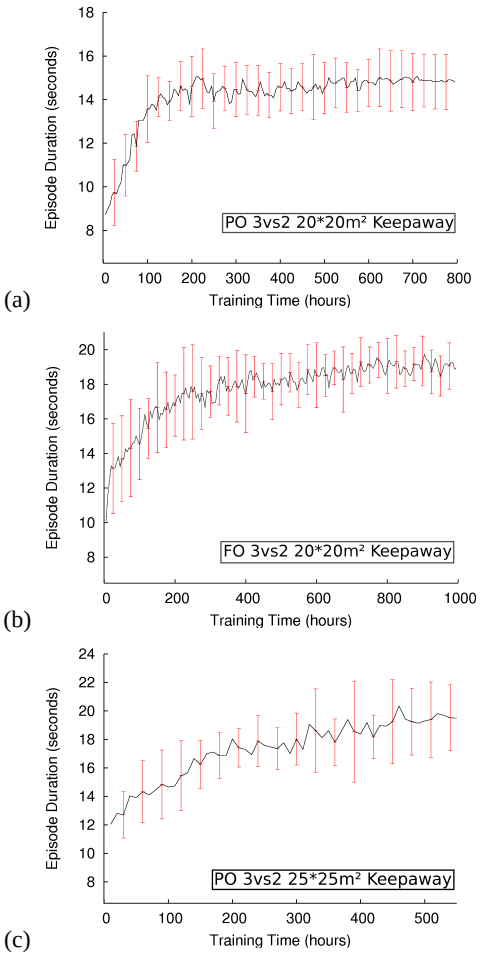

Figure 3: Average episode duration of EANT in: (a) partially observable 3vs2 Keepaway on a $20m \times 20m$ field (averaged over 8 independent runs), (b) fully observable 3vs2 Keepaway on a $20m \times 20m$ field (averaged over 9 independent runs), and (c) partially observable 3vs2 Keepaway on a $25m \times 25m$ field (averaged over 6 independent runs).

5.3 Partially Observable 3vs2 Keepaway on a Larger Field

In order to assess if the policies learned by EANT would also benefit from a larger field, we set the fieldsize to $25m \times 25m$ and performed 6 runs of partially observable Keepaway. As can be seen in Figure 3c, the average episode duration increased significantly in comparison with the average episode duration on a $20m \times 20m$ field: on average, EANT evolves a policy with an approximate fitness of $19.4sec$ with a standard deviation of $2.2sec$ after $500h$ training time. We suspect that the increase of the average episode duration is due to the fact that the Keepers have more space on the field and that the evolved policies have learned to exploit this fact.

5.4 Comparison with other methods

Stone et al. [20] have used an application of episodic SMDP Sarsa (λ) with linear tile-coding function approximation and variable λ to learn higher-level decisions in the keepaway benchmark. They have reported that their agents learned policies that significantly outperformed a range of benchmark policies as well as poli-

	EANT			NEAT			Sarsa(λ)		
	MED	TT	SD	MED	TT	SD	MED	TT	SD
PO 3vs2 $20m \times 20m$	14.9	≈ 200	± 1.25	14.1	800	$\approx \pm 1.75$	12.5	50	$\approx \pm 0.1$
FO 3vs2 $20m \times 20m$	19.2	≈ 600	± 1.16	15.5	800	$\approx \pm 1.00$	17.6	50	$\approx \pm 0.1$

Figure 4: Summarized performance of three different reinforcement learning methods (EANT, NEAT, and Sarsa(λ)) in different versions of Keepaway. Depicted are the maximal mean episode duration (MED), the training time required for reaching this optimum (TT), and the standard deviation over the different runs (SD). MED and SD are given in seconds, and TT in hours. Results for NEAT and Sarsa(λ) are obtained from Taylor et al. [22] and Whiteson et al. [25] (exact values based on personal communication).

cies learned with Q-learning[1]. Recently, Taylor et al. [22] and Whiteson et al. [25] have given a detailed empirical comparison between a variant of SMDP Sarsa (λ) [21] and the neuroevolution method NEAT [18] in the Keepaway benchmark. The results they obtained are summarized (along with the results of EANT) in Table 4.

They found that in general NEAT learns better policies than Sarsa in PO Keepaway, though it requires many more evaluations to do so. Moreover, they found that Sarsa learns better policies in FO Keepaway and NEAT learns faster when the task is deterministic (i. e. the start state is fixed and neither sensors nor actuators are influenced by noise). As can be seen, EANT achieves better results than NEAT and Sarsa in both PO Keepaway[2] and FO Keepaway[3]. Furthermore, it converges to good solutions with less training time than NEAT. These results reinforce results of Siebel et al. on a Visual Servoing Task [17], where EANT also performed better than NEAT. A possible explanation is that the structural mutation operations of NEAT are more fine-grained, and thus NEAT would require more mutations to reach a topology with sufficient complexity. An other explanation might be the explicit separation of the evolution into exploitation and exploration phases.

6. DETAILED ANALYSIS

The previous section has shown that EANT is able to learn good policies for the Keepaway problem. However, these results give no indication of the components of EANT that are and are not necessary to enable this kind of learning facility. In this section, we analyze the contribution of different parts of EANT to the overall learning performance. All results presented were obtained in the standard setting for Keepaway (i. e. partially observable 3vs2 Keepaway on a $20m \times 20m$ field).

6.1 Structural exploration

What distinguishes neuroevolutionary approaches such as EANT from conventional artificial neural network learning systems is that the topology of the network does not need to be fixed by the designer, but is generated and optimized by the system itself. It is worth considering the question of what performance a system can achieve that uses a means of weight optimization similar to EANT, but does not search the space of network structures (i. e. does not explore). To answer this question, we deactivated the structural mutation and crossover operators and applied this impaired system to the Keepaway task. The results of three independent runs are depicted in Figure 5a. The mean episode duration in partially observable 3vs2 Keepaway ($11.1 \pm 0.83 sec$ after $700h$, averaged over 3 independent runs) is significantly worse ($p < 0.015$) than those of the full system discussed in Section 5.1, indicating that structural exploration significantly improves the performance of EANT.

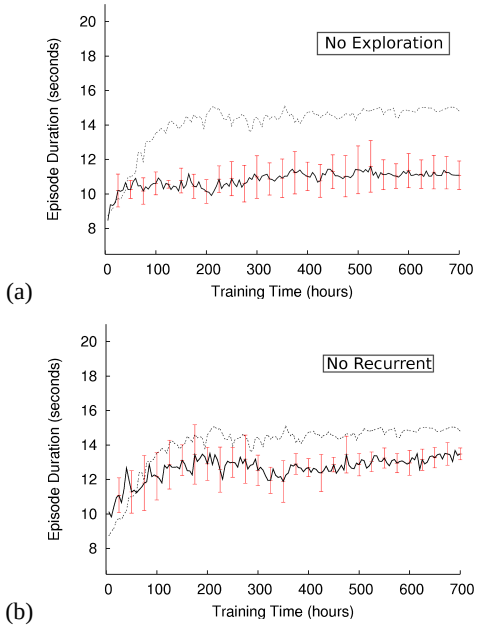

Figure 5: Average episode duration of impaired versions of EANT in partially observable 3vs2 Keepaway: (a) results of EANT when structural exploration is disabled, (b) results of EANT when the genotype is not allowed to contain recurrent jumpers. The results are significantly worse than the results obtained by the full system indicated by the dashed line.

6.2 Recurrent jumpers

Since the policy is represented as an ANN with recurrent connections, which give a neuron n_i at time step t the output of a neuron n_j at time step $t-1$ (possibly $n_i = n_j$), the action a_t chosen by the policy at time t is not necessarily based only on the the state s_t but can also be based on s_{t-r} for $r > 0$. This should not provide any advantage to system when used with an MDP. However, as the authors of the benchmark state, Keepaway is not truly Markovian but only nearly since the velocities of the players are not included in the state and the sensors are noisy. Thus, one might expect that a system which has some kind of a memory might perform better than one without memory. In order to analyze whether EANT's ability to evolve ANN's with recurrent connections actually improves its performance in non-Markovian environments, we deactivated the

[1]The absolute values of episode duration they reported are not directly comparable with the results reported below since Stone et al. used an older version of the benchmark.

[2]This result is statistically significant for the comparison to Sarsa ($p \leq 0.0005$).

[3]This result is statistically significant for the comparison to both, Sarsa ($p < 0.002$) and NEAT ($p < 10^{-5}$).

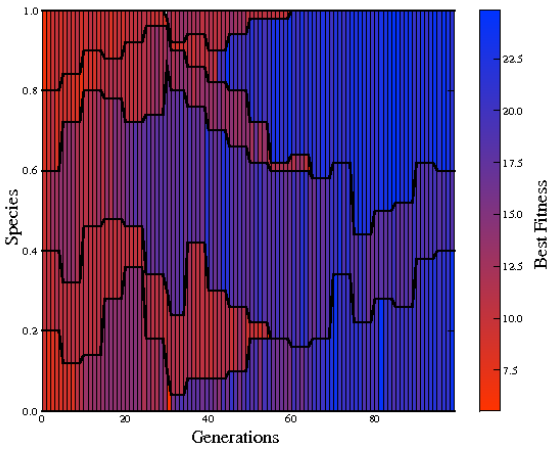

Figure 6: Development of the species of a population during one evolution. The width of a species (in y-direction) indicates the relative size of this species while the color represents the fitness of the best individual of this species.

possibility of evolving recurrent connections and tested this system in the Keepaway task. The results are depicted in Figure 5b. The mean episode duration of the impaired system in partially observable 3vs2 Keepaway ($13.48 \pm 0.35sec$ after $700h$, averaged over 5 independent runs) is significantly worse ($p < 0.05$) than those of the full system discussed in Section 5.1, indicating that recurrent jumpers significantly improve the performance of EANT. By contrast, in the early phase of the evolution, the impaired version of EANT performs better. We suspect that this is due to the higher complexity of finding good weights for networks with recurrent jumpers.

6.3 Speciation

The main purpose of speciation is to protect individuals with a novel topology from immediately extincting (compare Section 3.3.3). Figure 6 shows the development of species during the progress of one sample evolution. As can be seen, even species which have initially only individuals with low fitness survive for many generations. This is due to the usage of fitness sharing, which helps small species to survive. Interestingly, the two species which obtained the best performance at the end (after approximately 100 generations), were nearly extincted after 30 generations.

6.4 Fitness sharing

The purpose of fitness sharing (see Section 3.3.3) is to ensure that the population retains a sufficient amount of diversity during the progression of the evolution, i. e. to avoid that one species takes over the whole population. We define the "amount of diversity" of a population P_c as the average pairwise distance between all individuals in a population: $div(P_c) = \frac{1}{|P_c|^2 - |P_c|} \sum_{g_i, g_j \in P_c, g_i \neq g_j} d(g_i, g_j)$, where $d()$ is the genotypic distance given in Section 3.3.3. If fitness sharing is disabled, the diversity of a population decreases from an initial value of approx. 0.7 to a diversity of approx. 0.2 after $800h$ training time (see Figure 7a). In contrast, when fitness sharing is enabled, the diversity decreases slower and maintains a value of approx. 0.45 (averaged over 4 independent runs) after $800h$ training time. The increased diversity allows EANT to cover a larger part of the search space simultaneously.

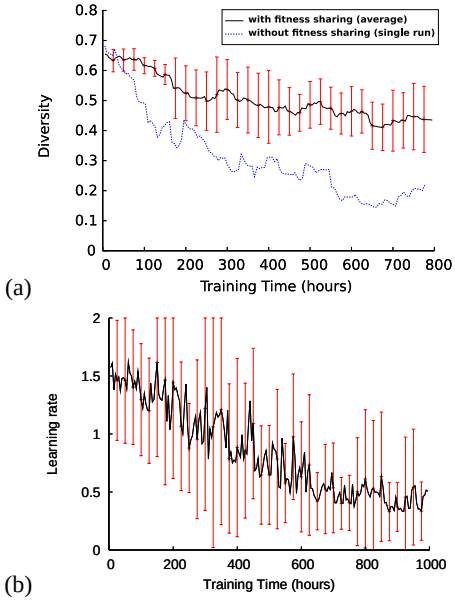

Figure 7: (a) Effect of fitness sharing on the diversity of a population. Without fitness sharing, the diversity decreases continuously to a very small value. In contrast, with fitness sharing, the diversity maintains a value above 0.45 on average. (b) Development of the learning rate during the evolution. Plotted data is an average of 8 independent runs.

6.5 Self-adaptation of learning rate

In EANT, each linear genome has a "strategy parameter" learning rate, which controls how strongly the weights are affected by the parametric mutation operator. This parameter is set initially to 1.0 for all individuals. During the course of the evolution, the learning rate itself is modified randomly by the parametric mutation operator (compare Section 3.2.1). Since this modification is unbiased, one might expect that the average learning rate of the individuals remains nearly constant and close to 1.0 during the course of evolution. However, Figure 7b shows that this is not the case: depicted is the development of the learning rate of the champion (i. e. the fittest individual), averaged over 8 independent runs in the fully observable 3vs2 Keepaway domain. As can be seen, the learning rate increases initially up to a value of 1.5. After $200h$ training time, the learning rate starts to decrease to a value of less than 0.5.

An interpretation of this behavior is as follows: initially, the values of the weights are far away from their optimal value. Thus, these weights have to be modified significantly to achieve a good overall performance of the network. Genomes with larger learning rate can do bigger modifications of their weights in one application of the parametric mutation and because of that, their offspring has a better chance to approach a good set of weights. Hence, their offspring has a better chance to survive. This leads to an increased density of genomes with large learning rates. After $200h$ of training time, the genomes have increased their performance significantly (compare Figure 3b). This indicates that their weights are now closer to an optimal setting. The initial advantage of genomes with large learning rate starts now to turn into a disadvantage: since the modifications of the parametric mutation operator affect their weights more drastically, the probability that their offspring leaves the local optimum weight region is greater than for genomes with lower learning rates. Hence, the average learning rate decreases as

the system's performance nears the optimum. This behavior exhibits the same self-adaptation of learning rates that is found in Evolutionary Strategies [15].

7. CONCLUSION AND OUTLOOK

In this paper, we have shown that the neuroevolutionary method EANT can perform better than the methods which have been tested previously in the Keepaway benchmark. We have shown that fitness sharing guarantees that a sufficient amount of diversity is retained in the population. Furthermore, we have shown that the ability of EANT to explore the space of network topologies is crucial for its overall performance and that networks with recurrent connections can achieve a better performance in non-MDP environments. In the future, it would be interesting to analyze EANT's performance on more complex Keepaway tasks such as 4vs3, or 5vs4 Keepaway. Furthermore, a combination of a neuroevolutionary method like EANT with a TD method like Q-Learning (as proposed by Whiteson [24]), or a genetic encoding utilizing an explicit or implicit embryogeny could be tested and analyzed in the Keepaway domain. With regard to EANT, experiments could be conducted that analyze how crucial separate exploitation and exploration phases are, and if a specific choice of structural mutation operator improves the rate at which the system learns.

8. ADDITIONAL AUTHORS

9. REFERENCES

[1] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65, 1994.

[2] J. E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[3] P. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 35–43, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[4] J. C. Bongard and R. Pfeifer. Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836, 2001.

[5] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning*, 2006.

[6] F. Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Ecole Normale Superieure de Lyon, Laboratoire de l'Informatique du Parallelisme, France, January 1994.

[7] Y. Kassahun. *Towards a Unified Approach to Learning and Adaptation*. PhD thesis, Technical Report 0602, Institute of Computer Science and Applied Mathematics, Christian-Albrechts University, Kiel, Germany, Feb 2006.

[8] Y. Kassahun, M. Edgington, J. H. Metzen, G. Sommer, and F. Kirchner. A common genetic encoding for both direct and indirect encodings of networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1029–1036, 2007.

[9] Y. Kassahun, J. H. Metzen, J. de Gea, M. Edgington, and F. Kirchner. A general framework for encoding and evolving neural networks. In *Proceedings of the 30th Annual German Conference on Artificial Intelligence*, pages 205–219, 9 2007.

[10] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

[11] A. Lindenmayer. Mathematical models for cellular interactions in development, parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[12] S. Nolfi and D. Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Massachusetts, London, 2000.

[13] S. Nolfi and D. Parisi. Growing neural networks. Technical Report PCIA-91-15, Institute of Psychology, Rome, 1991.

[14] J. Reisinger and R. Miikkulainen. Acquiring evolvability through adaptive representations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1045–1052, New York, NY, USA, 2007. ACM Press.

[15] H.-P. P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[16] B. Sendhoff and M. Kreutz. Variable encoding of modular neural networks for time series prediction. In *Congress on Evolutionary Computation*, pages 259–266, 1999.

[17] N. Siebel, J. Krause, and G. Sommer. Efficient learning of neural networks with evolutionary algorithms. In *Proceedings of the 29th German Symposium for Pattern Recognition*, pages 466–475, 2007.

[18] K. O. Stanley. *Efficient Evolution of Neural Networks through Complexification*. PhD thesis, Artificial Intelligence Laboratory. The University of Texas at Austin., Austin, USA, Aug 2004.

[19] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer Verlag, Berlin, 2006.

[20] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer Keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[21] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Massachusetts, London, 1998.

[22] M. E. Taylor, S. Whiteson, and P. Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1321–1328, 2006.

[23] J. Vaario, A. Onitsuka, and K. Shimohara. Formation of neural structures. In *Proceedings of the Fourth European Conference on Articial Life*, pages 214–223, 1997.

[24] S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006.

[25] S. Whiteson, M. E. Taylor, and P. Stone. Empirical studies in action selection for reinforcement learning. *Adaptive Behavior*, 15(1):33–50, March 2007.

[26] D. Whitley, S. Dominic, R. Das, and C. W. Anderson. Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13:259–284, 1993.

[27] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.