Evolving Neural Network Controllers for a Team of Self-organizing Robots

István Fehérvári and Wilfried Elmenreich
Mobile Systems Group/Lakeside Labs
Institute for Networked and Embedded Systems
University of Klagenfurt, Klagenfurt, Austria

Abstract

Self-organizing systems obtain a global system behavior via typically simple local interactions among a number of components or agents, respectively. The emergent service often displays properties like adaptability, robustness, and scalability, which makes the self-organizing paradigm interesting for technical applications like cooperative autonomous robots. The behavior for the local interactions is usually simple, but it is often difficult to define the right set of interaction rules in order to achieve a desired global behavior. In this paper we describe a novel design approach using an evolutionary algorithm and artificial neural networks to automatize the part of the design process that requires most of the effort. A simulated robot soccer game was implemented to test and evaluate the proposed method. A new approach in evolving competitive behavior is also introduced using Swiss System instead of the full tournament to cut down the number of necessary simulations.

Keywords: Artificial neural networks, Cooperative Robots, Self-organizing Systems, Evolutionary algorithm

1 Introduction

The concept of systems consisting of multiple autonomous mobile robots is attractive for several reasons [1]: Multiple cooperative robots might be able to achieve a task with better performance or with lower cost. Moreover, loosely coupled distributed systems tend to be more robust, yet more flexible than a single powerful robot performing the same task. A benefit of the collaborative interaction of mobile robots can be an emergent service, i.e., a progressive result that is more than the sum of the individual efforts [2]. A swarm of robots can thus build a self-organizing system [3].

The continuous technical development in robotics during the last decades has provided us the hardware for swarms of small, cheap autonomous devices [4, 5, 6]. However, designing the behavior and interactions among the robots remains a very complex task. Using a standard top-down design approach with fixed task decompositions and allocation typically leads to systems working only for a small set of parameters. On the other hand, effects like changing environments or breakdowns and faults of hardware require a robust and flexible solution that provides a useful service for many possible system states.

An alternative to the classical design approach is to organize the robots as a self-organizing system performing the intended task. Thus, the robots achieve a global system behavior via simple local interactions without centralized control [7]. As shown by many examples in nature, simple rules for interactions can emerge to quite complex behavior while being scalable and robust against disturbances and failures. This would allow for simple control systems like for example having a small Artificial Neural Network (ANN) on the particular robots.

Unfortunately, there is yet no straight-forward way for the design of these rules so that the overall system shows the desired properties. Typically, the emergent service is really hard, or even impossible to predict. Thus, finding a set of rules that causes the overall system to exhibit the desired properties presents a great challenge to the system designers. The main problem is that a small change of a parameter might lead to unexpected and even counter-intuitive results [8, 9].

To design a self-organizing system with the desired emergent behavior, it is crucial to find local rules for the behavior of the system's components (agents) that generate the intended behavior at system scale. In many cases, this is done by a sumptuous trial and error process which in case of systems with high complexity is not efficient or even unfeasible. Parameter-intensive systems also suffer from the unpredictability of the results due to unexpected dependencies between parameters.

In this paper we discuss the application of evolutionary methods for designing an ANN-based control system for a team of self-organizing robots. In particular, we tackle

the interface design between neural controller and robot and elaborate the particular influence of fitness function parameters on the results. As a case study for the approach, we describe the evolution of the neural control program for simulated soccer robots.

The paper is structured as follows: In the next section we give an overview about self-organization and systems presenting the background of this paper will be given. Then the design steps as a general approach with the evolutionary algorithm will be described. Section 4 focuses on the practical evaluation of the approach, presenting the setup of the robot soccer simulation while Section 5 shows and explains the acquired results. Related work is discussed in Section 6. This paper is concluded in Section 7.

2 Self-organizing Systems

The concept of self-organizing systems (SOS) was first introduced by W. Ross Ashby [10] in 1947 referring to pattern-formation processes taking place within the system by the cooperative behavior of its individuals. These could best be described by the way they achieve their order and structure without having any external directing influences. There are several definitions for SOS [11], the following was formulated on the Lakeside Research Days'08 [7]:
"*A self-organizing system (SOS) is a set of entities that obtains global system behavior via local interactions without centralized control.*"
An example could be a team of workers acting on their own following a mutual understanding. If there was any external influence like a common blueprint or a boss giving orders it would result in no self-organization. Many examples of SOS can also be observed in nature, e.g. a school of fish where each individual has knowledge only about its neighbors and having no leader amongst them. The key part is the communication between the individuals; the way they satisfy their own goal such as getting close, but not too close to other fish in the school while trying to find food in the water.

Furthermore it is important to note that the emergent property cannot be understood by simply examining the system's components in isolation, but requires the consideration of the interactions among the components. This interaction is not necessarily direct, it can be indirect as well when one component modifies the environment which then influences other components [12]. This presents a continuous mixture of positive and negative feedback in the behavior.

By describing self-organizing systems the following advantages can be observed: These systems are often very robust against external disturbances; it is clear that a failure of one component rarely results in a full collapse. Also adaptability and scalability can be noticed meaning a dynamical behavior and a flexibility in the number of components. It is also important to note that usually, once the local rules of the SOS are found, the implementation takes less effort compared to a centralized solution. These properties make SOS an interesting, though difficult to design, option for the decentralized control of complex systems. Although there are some ideas for designing such systems, there is no general methodology yet explaining how this should be done.

3 Evolutionary Approach to Design SOS

This section describes the proposed method giving SOS design engineers a tool. The process starts by defining the main goals: what are the expectations from our system. The next step is to build an evolvable representation of local behavior. Our approach uses an evolutionary algorithm to explore the solution space. Therefore, the evolvability of the representation is required, which means that operators like mutation and/or recombination must be defined on the representation. Instead of using a standard representation like a bit string as in genetic algorithms, the applied algorithm employs mutation and recombination functions which are representation-specific. We have implemented a Java program called FREVO[1] which is an open-source framework for evolutionary design. It identifies and separates the key components, such as the problem definition, candidate representation and the optimization method. As depicted in Fig. 1, our framework supports different representations, where each must embed specific functions for mutation and combinations of candidates. The advantage of this approach is that the search space can be reduced since operations which likely produce unfeasible solutions are filtered. On the other hand, there is an increased effort for the implementation of a new representation by adding the specific mutation/recombination functions. Usually, control software is written in programming languages (JAVA, C, etc.) which are inappropriate for phenotypical mutation and recombination operators. One notable exception could be the LISP programming language which is used for the representation of an evolvable algorithm in [13]. Unlike standard programs, artificial neural networks (ANNs) or representations based on fuzzy rules are qualified for this task. Structure and setup of this representation is also a question; it can be trained by the evolutionary algorithm or defined *a priori*.

In case of ANNs reinforced learning is needed, since

[1] http://www.frevotool.tk

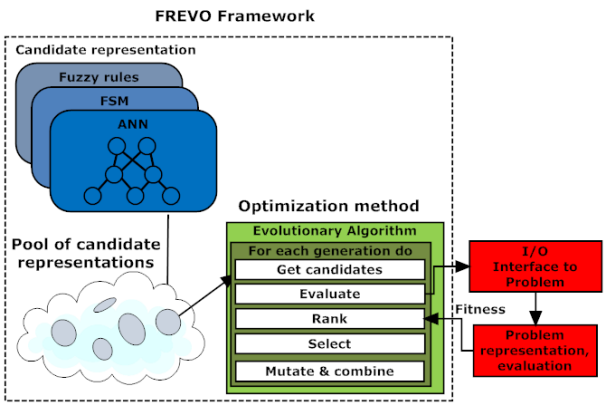

Figure 1: Components of the FREVO framework: The *agent representation* is optimized by the *evolutionary algorithm* to maximize the fitness for the given *problem*

we have to deal with belated rewards we get after a simulation of many steps of the revised ANN. Thus, the standard back propagation algorithm cannot be applied to program the ANN's weights. At this point we need our goals to be formulated as rewards for reinforcing the candidates. To make the learning process smoother we propose a step by step approach decomposing the overall goal into smaller achievements weighted according to their significance. A typical example would be an object manipulation task for a robot where the three subtasks would be: finding, grabbing and then manipulating the object. In the next chapter an example of this approach will be given.

With a simulation environment acting as a playground, the evolutionary algorithm can start evolving the possible candidates. Typically, a fitness value can be deduced from the simulation results. This fitness value is then used in the evolutionary algorithm to decide on the fittest individuals. An example for such a fitness value could be the throughput of a given setup of a wireless network. In many cases an absolute fitness value cannot be assigned especially when a competitive emergent behavior is expected. In order to get a ranking of the individuals it is necessary to play a tournament among the candidates of a population. In a native approach, the number of pairings equals $n(n-1)/2$; n being the number of individuals in a population. In case of long simulations the time can be cut effectively by using the Swiss System, a pairing system used to organize (chess) tournaments which yield a ranking with a minimum number of pairings [14] instead of full tournament. Detailed description can be found in the next section.

4 Case Study

As a case study, we have defined the problem of teaching soccer to a team of autonomous robots in a 2D environment similar to the official RoboCup Simulation League. This problem provides a rich testbed domain for the study of control, coordination and cooperation issues described in [15] [16]. We also presented some early results for this problem in [17]. In the following we give a brief description of the actual problem of simulated robot soccer. Then, we present the representations of the particular elements according to the components depicted in Figure 1.

4.1 Problem Description

The problem is evaluated via a robot soccer simulator with simple physics, similar to the official simulator used for the RoboCup simulation league. In contrast to the official simulator, ours does not include the roles of a referee or goalkeeper and there is a simulated boundary around the field, which avoids situations where the ball goes out of bounds. The main change with respect to our approach is that our simulation does not run in real time (except for a built-in demonstration mode), but as a discrete event simulation that runs with maximum computation speed. This greatly reduces the actual time for performing the evolution. The chosen problem consists to the class of competitive evaluation, i.e., we can only compare the relative fitness of two candidate solutions by simulating a match between them.

A simulation run consists of initially placing a configurable number of soccer players (typically 2x11) on a field and to simulate a game where each player can accelerate with a given strength towards a given direction and, if being close enough to the ball, can kick the ball with a given strength towards a given direction. One game lasts for 300 steps which yields 60 real-time seconds.

4.2 Optimization method

We used an evolutionary algorithm to evolve the controllers of the soccer players. The implementation of the optimization method is based on the one presented by Elmenreich and Klingler in [18]. The size of the population was set to 60 and the length of the simulation was fixed at 500 generations. The parameters of the genetic operators can be seen in Table 1.

4.3 Candidate Representations

The candidates for the evolutionary algorithm were realized as ANNs. Training has been applied to optimize the weights and biases of the neural network. We tested

Population size	60
Number of generations	500
Percentage of elite selection	15
Percentage of mutations	40
Percentage of crossover	30
Percentage of randomly created offsprings	5
Percentage of randomly selecting an offspring from previous generation	10

Table 1: Parameters of the evolutionary algorithms

two different candidate representations in our case study, namely a fully connected ANN and a three-layered ANN.

The fully connected network is a time-discrete, recurrent artificial neural network. Each neuron is connected to every other neuron and itself via several input connectors. Each connection is assigned a weight that is a floating value and each neuron is assigned a bias. The problem requires 16 inputs and 4 outputs. Additional "hidden" neurons are added in order to increase the expressiveness and the number of representable states.

The three-layered network only provides forward connections from the input nodes (the input layer) to the nodes in the hidden layer and forward connections from the hidden layer to the output layer.

In most cases, three-layered networks are employed for ANN applications, since they can be programmed via back propagation. However, our setup provides only belated rewards, that is feedback after a simulation involving many different actions of the ANN controller. The evolutionary algorithm works with both representations, so we can easily include the fully connected network.

The implementation of both types is almost identical, the only difference is that the three-layered network only features a subset of connections per neuron. At each step, each neuron i builds the sum over its bias b_i and its incoming connection weights $w_j i$ multiplied by the current outputs of the neurons $j = 1, 2, ..., n$ feeding the connections. Weights can be any real number, thus have either an excitatory or inhibitory effect. The output of the neuron for step $k + 1$ is calculated by applying an activation function F:

$$o_i(k+1) = F(\sum_{j=0}^{n} w_{ji} o_j(k) + b_i)$$

where F is a simple linear threshold function

$$F(x) = \begin{cases} -1.0 & \text{if } x \leq -1.0 \\ x & \text{if } -1.0 < x < 1.0 \\ 1.0 & \text{if } x \geq 1.0 \end{cases}$$

As described below in detail, the output of the output neurons is further scaled to match the input range of the actuators.

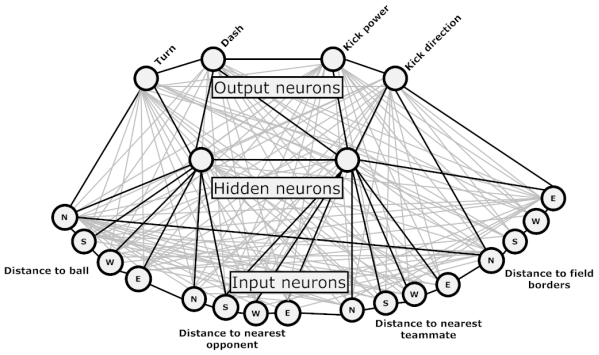

Figure 2: A possible wiring of the neural network showing the groups of inputs, outputs and hidden neurons. Connections with stronger weight are indicated with bold lines while ones with lower weight are colored with grey.

4.4 I/O interface between Simulation and Controllers

In the case study, the information passed to the simulation is predefined; it consist of the strength and direction for the players' move and, in case the player can kick the ball, the strength and direction of the kick. The information provided by the simulation is also determined; it consists of the position of the ball (if visible), a list of visible teammates, a list of visible opponent players, and information about the distance to the field's (upper, lower, left, right) border. The position of the goal is given indirectly by combining the distance to the borders with the knowledge that the goal resides in the middle of the sideborders.

However, there are different ways to pass this information into the ANN. In general, the ANN will have a set of so-called input neurons, which activate their output according to a given input from external sources. Respectively, a number of output neurons is used to export information from the network. Finally, some unspecified or hidden neurons are added. All neurons are interconnected by directed weights, which are evolved in the framework (See Figure 2).

Since the number of neurons defines the search space, the number of neurons should not become too large. On the other hand, input neurons should be defined in a way that their result is easily interpretable by the ANN. For example in [18], we modeled a distance sensor that is periodically changing its orientation via several input neurons, each representing the input for a particular orientation.

In the robot soccer example, the inputs are arranged

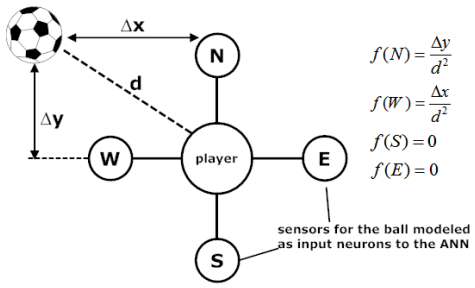

Figure 3: A group of input neurons detecting the ball

i	P_i	W_i
p	field distribution	10^0
bd	distance to the ball	10^3
k	number of kicks	$2 \cdot 10^4$
fk	number of false kicks (ball is kicked out of bounds)	10^4
bg	ball distance to the opponent's goal	10^5
s	number of scores	$4 \cdot 10^6$

Table 2: Parameters of the fitness function

in groups of four neurons. Each group is responsible for communicating the detection of a particular object class (ball, teammate, opponent, border) and consists of a *north*, *south*, *east* and *west* neuron.

If the nearest object of that class is in a particular quadrant, lets say north-west, then the north neuron and the west neuron are activated inversely proportionally to the components of the vector to the object. So, if in our example the ball is towards the north-north-west, the north neuron gets a high activation and the west neuron a moderate one (See Figure 3).

For the output neurons, we tested two setups: in the first setup, the outputs are scaled to $[-100\%, 100\%]$ and $[-180, +180]$, respectively and interpreted as polar coordinates for the move and kick vectors. The second setup interprets the neurons as being the x and y components of a vector in cartesian coordinates. In general, both approaches are expected to work, since they transport exactly the same information and the ANN will be evolved to adjust to the given representation.

4.5 Fitness evaluation

Typically, when the task is more complex the definition of the fitness function is not trivial. In the case of a soccer game the primary aim is to train teams scoring the most goals during the given time interval. However, this problem is far too complicated for teams, initially composed of random ANN controllers, to expect improvement over generations just by rewarding them by the final number of goals they score in each game. The idea here is to decompose the overall goal into smaller achievements (so-called guidelines) and let the teams fulfill them one after the other. This method tries to ensure a smooth learning process assuming some preliminary knowledge or ideas about the solution. The guidelines are assigned a weight to setup a hierarchical order. It means a task with smaller weight is less important, but will most likely be accomplished before another tasks with higher value. This is because the second one is too complex to be achieved without learning the first one. Figure 4 shows the applied tasks in their respective order in our simulation.

At the beginning of the training we wanted the teams to learn that a good distribution on the field might lead to good overall play. Therefore, we introduced the first guideline (field distribution). It was implemented by defining 64 evenly distributed checkpoints on the field and counting the number of controlled points every 5 seconds for both teams. A point is controlled by a team if it has the nearest player to this point. The accumulated points are added to the final fitness value. The second guideline was an advice for the teams to move their players closer to the ball. The distance of the nearest player for both teams to the ball is measured and compared every 4 seconds. The team having a player closer to the ball earns one point. At the end of the game this point is weighted and also added to the final fitness. The number of kicks is also counted with a weight however only the first 10 kicks are taken into account to create an upper bound and to prevent dead team strategies where they only pass the ball back and forth. Concerning the kicking direction the ball distance to the opponents goal is also measured and calculated every 2 game seconds in the same manner as guideline two. The highest weight is assigned to the number of scores, being more significant than the other fitness components. Therefore, we define the fitness function as the following equation:

$$F = W_p P_p + W_{bd} P_{bd} + (W_k P_k - W_{fk} P_{fk}) + W_{bg} P_{bg} + W_s P_s$$

where W and P stand for the weights and the points respectively. Table 2 explains the corresponding indexes and values.

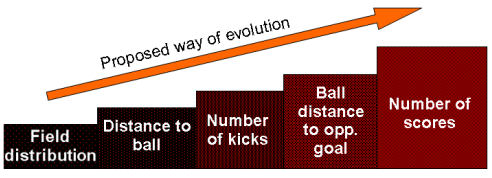

Figure 4: Weighted fitness

4.6 Tournament Ranking with Swiss System

Evolving competitive team behavior is a good example where one cannot assign a simple absolute fitness value. To rank the teams one solution is to play a tournament among the candidates in each generation (assuming one population with n candidates). A full tournament would mean $n(n-1)/2$ number of pairings when n is the number of entities in the population. In case a simulation run takes too much time or a high number of generations is needed, this approach can be very ineffective. For example, a population of 50 individuals would require 1225 runs for each generation. The proposed solution tries to minimize the number of necessary pairing using Swiss System style tournament [19]. It reduces the required number to $\lceil \log_2 n \rceil \frac{n}{2}$ which is in the mentioned case only 150 games (see Figure 5). Inspired by the official FIDE[2] rules for chess tournaments we established the following system:

In each game the winner gets two points, loser gets zero, in case of a draw both get one point. After the first round players are placed in groups according to their score (winners in group "2", those who drew in group "1", and losers in group "0"). The aim is to ensure that players with the same score play against each other. Since the number of perfect scores is cut in half each round, it does not take long until there is only one player left with a perfect score. The actual number of rounds needed is $\log_2 n$ to be able to handle n teams. In chess tournaments there are usually many draws, so more players can be handled (a 5 round event can usually determine a clear winner for a section of at least 40 players, possibly more), although in our simulation a draw is very unlikely. To avoid early games between elite selected entities, the first round is not randomized but cut into two halves where the first half, consisting of teams which have performed well so far, is playing against the second half.

The drawback of the Swiss system is that it is only designed to determine a clear winner in just a few rounds. Regarding other players, we have little information about their correct ranking. For example, there could be many players with 3-2 scores and it is hard to say which player is better than the other, or whether a player with 3.5 points is better than a player with 3 points. To help determine the order of finish, a tiebreak method has been implemented. In order to decide on the ranking for players having the same score, we used a method developed by Bruno Buchholz [20]. There the score of the players' opponents is summed up thus favoring those who have confronted better opponents. In case it is still undecided the sum is extended by the points of those opponents who have lost against the player. This uncertainty in the ranking could

[2]http://www.fide.com

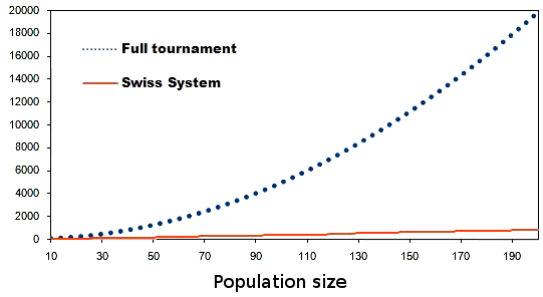

Figure 5: Total number of games in full tournament and Swiss System

cause problems in the evolutionary algorithm when selecting entities for survival to the next generation. In our case elite selection was 15% while the Swiss System ensures only the first and last position to be ranked correctly, thus the position of all other players carries also some obscurity. After observing this effect in our simulation we came to the conclusion that having a somewhat imprecise selection among the top players slows down the process just a little or not at all. To select entities for survival we used a roulette wheel selection where the probability being selected is directly proportional to the fitness, in our case the ranking of the Swiss System. Since this approach already carries some randomization some more uncertainty did not make a crucial impact.

5 Results

We ran several simulations evolving soccer teams. In particular, we varied

- the type of representation (fully connected or layered ANN),
- the number of hidden nodes (2, 4, or 6),
- the type of the interface between simulation and controllers.

We evolved each setting up to 500 generations. Unfortunately, there is no absolute fitness value for depicting the quality of an evolved result. Only relative comparisons of teams by matching them in a simulation are possible. For our evaluation, we picked the best result of every 20th generation. These "champions" have than been matched in a round-robin tournament against each other in order to determine if there is a constant evolution towards better gameplay and if one setting is performing better than the other.

We found out that the design of the interface between simulation and controllers is of major importance to the success of the evolutionary algorithm. The results

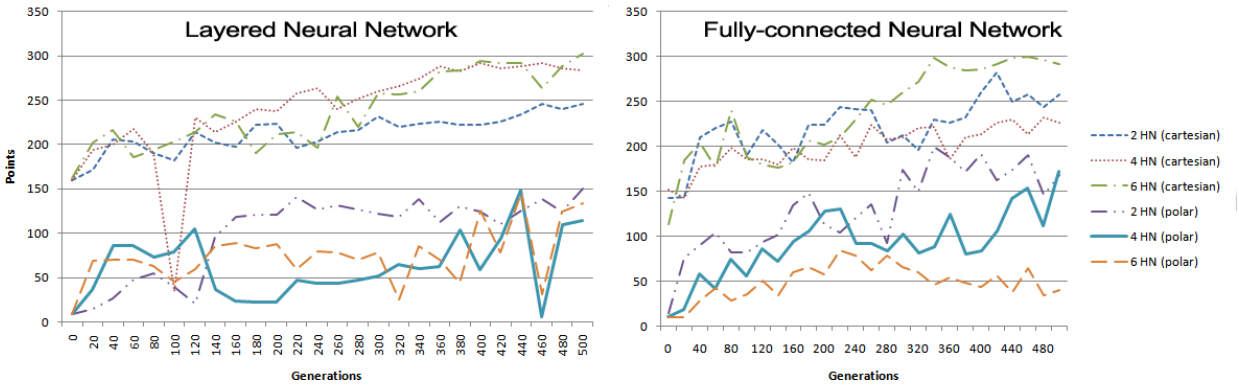

Figure 6: Tournament results of ANNs with different I/O interfaces

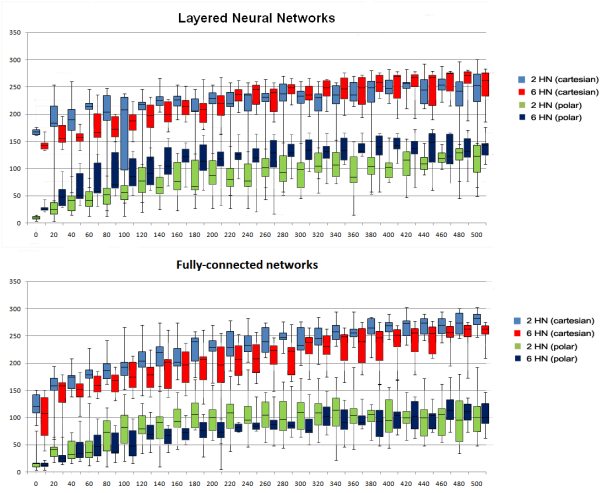

Figure 7: Box-and-whisker diagram of the repeated evaluation of different I/O models and different number of hidden neurons for layered and fully connected ANNs

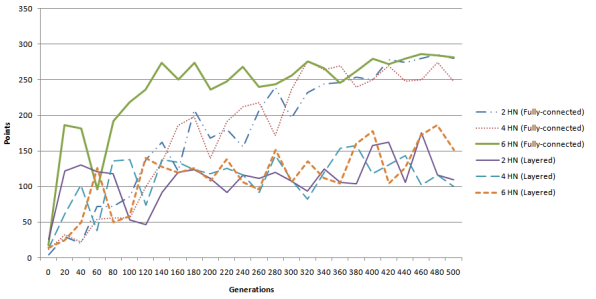

Figure 8: Tournament results of fully connected vs. layered ANNs with cartesian interface

showed that the selection of the interface between simulation and controllers has a significant influence on the speed of convergence and quality of the evolved solution. When the output neurons were interpreted as polar coordinates, the ANN controller needed to learn the coherent semantics of polar coordinates, and, probably, learn to emulate trigonometric functions. Figure 6 visualizes this observation for both, layered and fully connected ANNs. The figure depicts the results of a tournament of the above mentioned champions for various settings. Most curves are increasing over generations which depicts that the gameplay of the teams has been improved by the evolutionary algorithm.

As can be seen in the graphs, the systems using cartesian coordinates, even after several hundreds generations, are ranked lower than almost every other systems using cartesian coordinates. So, in this case, yielding output in polar coordinates posed a higher "cognitive complexity" for the system than yielding output in cartesian coordinates. Figure 7 shows a boxplot covering 10 different iterations of the evolutionary algorithms and also confirms that the cartesian coordinate I/O model is significantly superior to the polar coordinate model. This, however, may be specific for the chosen problem and may be different for other problems.

There was no significant effect of the number of hidden neurons on the overall performance or speed of convergence. This could be explained by the fact that a less complex ANN is already sufficient to learn the local interactions producing a competitive behavior.

Figure 8 compares the different versions using polar coordinates with each other and depicts that the fully connected ANNs have evolved faster and to a better gameplay than the layered networks. The box plot diagram in Figure 9 gives a statistic over 10 different runs of the evolutionary algorithm. While it confirms that all fully connected ANNs are typically better than the layered ANNs with 2 hidden neurons, it also shows that a layered ANN with enough neurons (that is 6 in that case) is in many it-

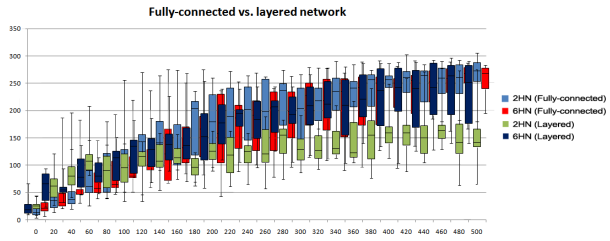

Figure 9: Box-and-whisker diagram of the repeated evaluation of fully connected vs. layered ANNs with cartesian interface

erations able to compensate for the lower number of connections.

Thus, a fully connected network with 6 hidden neurons and an I/O interface based on cartesian coordinates evolved 400 generations or more performed best according to the ability to win over others. Unfortunately, the quality and elegance of the result cannot be measured in this terms. By watching several games we observed the following behavior[3]:

- the player nearest to the ball runs to the ball
- other players (of the same team) in the vicinity of the ball also follow the ball, but they do usually not converge to the same spot; instead they keep spread out
- the player at the ball kicks it to a direction bringing it nearer to the opponent's goal
- players far from the ball spread out and build a defense mesh in front of their own goal
- players sometimes tend to stick to opponent players (man-marking)

Considering the relatively small size and complexity of the neural network controllers, the versatility of the emerging strategy is impressive. When both teams are well evolved, the ball is passed over several stations until the ball possession changes. Goals are scored roughly every few hundred simulation steps.

6 Related Work

In literature, only a few proposals for designing self-organizing systems can be found: First, a method proposed by Gershenson [21] introducing a notion of "friction" between two components as a utility to design the overall system using trial and error. Methods building on trials, even if they are improved by certain notions, often suffer from counter-intuitive interrelationships between local rules and emergent behavior.

Observing and learning from nature is also proven to be useful in several scenarios [22]. If an appropriate model exists and is available for study, top-down approaches can be very effective by applying the same phenomena. Conversely, bottom-up approaches adopt principles from nature and use on a fundamental basis [23].

There is also an imitation-based approach proposed by Auer [24] where the behavior of a hypothetical omniscient "perfect" agent is analyzed and mimicked in order to derive the local rules. A good example would be a perfect poker player who can see the hand of all other players and his decisions can be analyzed to create a relatively good normal player. The problem here and in all methods based on imitation is the limitation to cases where an appropriate example model is available.

Evolutionary algorithms have been applied in several ways to evolve ANNs. Yao [25] describes a general method for simultaneously evolving the weights of an ANN. Meeden [26] proposes a solution solely based on mutation and selection without crossovers. Floreano and Mondada [27] describe the evolving of a navigation system based on discrete-time recurrent networks. They successfully evolve a network for controlling a single Khepera robot. The work of Baldassarre et al. [28] shows evolving physically connected robots using ANN controllers. Sipper [29] shows the versatility of the approach by applying it to different game playing problems.

Nelson [30] describes the evolution of multiple robot controllers towards a team that plays "Capture the flag" against an opponent team of robots. This work applies the relative fitness concept as it is proposed in our approach. In contrast, Coelho et al. [31] evolve soccer teams evaluating against a control team. In particular they evolve separate ANNs for the different player behaviors (defense, middle and attack). In our approach, we evolve teams, where the same replicated ANN controls all the players. Based on inputs about situation and the behavior of the other teammates, a self-organizing process leads to a differentiation into the particular behavior types during runtime.

7 Conclusion and Future Work

We have described a method for evolving neural network controllers for a team of cooperative robots. Given an overall goal function, we evolve the particular weights and biases of the neural network controllers using an evolutionary algorithm. Thus, the neural network learns to interpret the sensory inputs, to control the robots actuators and to behave according to a strategy that is benefi-

[3] A video of the evolution of the gameplay can be found at `http://mobile.uni-klu.ac.at/demesos`

cial for the given task. The approach is very flexible and can be applied to a wide variety of problems but it depends on a sufficiently accurate simulation and a fitness function that provides the necessary gradients for the evolutionary algorithm.

In a case study, we have evolved a control behavior for simulated soccer robots to cooperatively win soccer games. After a few hundred generations, the players of a team adopt a useful behavior. In contrast to related work, the players were not evolved to *a priori* defined roles, like defender, midfielder or striker, but all have an instance of the same neural network controller. Still, during a game, different behavior of the players emerge based on their situations. Thus, similar to biological systems, the entities specify to different roles in a self-organizing way. Since the entities are identical, the system has a high robustness against failure of some of the entities.

We have examined the influence of various factors to the results. The most important factor was the design of the interface between neural network and sensors/actuators. Although an ANN could theoretically adopt to different representations of sensor/actuator interfaces, it was necessary to find an interface with low "cognitive complexity" for the ANN, which was in our case a simple cartesian representation of the sensors and intended robot movements. Furthermore, we analyzed the influence of using different sizes and types of ANNs. While the number of neurons had the smallest effect on the performance, the type of representation favored the fully connected network type.

In the future, we plan to assess the robustness and fault-tolerance of the generated solutions. Furthermore, our system is open to changes in the representation (e.g., using different controller types), the optimization method, and the problem definition (e.g., apply the approach to different problem domains).

Acknowledgment

This work was supported by the European Regional Development Fund and the Carinthian Economic Promotion Fund (contract KWF 20214|18128|26673) within the Lakeside Labs project DEMESOS. We would like to thank Herwig Guggi and Kornelia Lienbacher for the constructive comments on an earlier version of this paper.

References

[1] Y. Uny Cao, A. S. Fukunaga, and A. B. Khang. Cooperative mobile robots: Antecedents and directions. *Autonomous Robots*, 4:1–23, 1997.

[2] S. Johnson. *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. Scribner, 2002.

[3] D. Floreano and S. Nolfi. *The role of self-organization for the synthesis and the understanding of behavioral systems*, chapter 1, pages 1–18. MIT Press: Cambridge, 2000.

[4] G. Novak. Roboter soccer: An example for autonomous mobile cooperating robots. In *Proceedings of the First Workshop on Intelligent Solutions for Embedded Systems (WISES'03)*, pages 107–118, Vienna, Austria, 2003.

[5] S. Bergbreiter and K. S. J. Pister. Design of an autonomous jumping microrobot. In *IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007.

[6] J. Roberts, T. Stirling, J. Zufferey, and D. Floreano. Quadrotor using minimal sensing for autonomous indoor flight. In *European Micro Air Vehicle Conference and Flight Competition (EMAV'07)*, 2007.

[7] W. Elmenreich and H. de Meer. Self-organizing networked systems for technical applications: A discussion on open issues. In J.P.G. Sterbenz. K.A. Hummel, editor, *Proceedings of the Third International Workshop on Self-Organizing Systems*, pages 1–9. Springer Verlag, 2008.

[8] M. Resnick. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds (Complex Adaptive Systems)*. The MIT Press, January 1997.

[9] I. Harvey, E. Di Paolo, M. Quinn, and E. Tuci. Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, 11:79–98, 2005.

[10] W. R. Ashby. Principles of the self-organizing dynamic system. *Journal of General Psychology*, 37(3):125 – 128, 1947.

[11] S. Camazine, J.-L.Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[12] E. Bonabeau. Editor's introduction: Stigmergy. *Special Issue of Artificial Life on Stigmergy*, 5(2):95–96, 1999.

[13] J. R. Koza, editor. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1st edition, December 1992.

[14] T. Just and D. B. Burg, editors. *U.S. Chess Federation's official rules of chess*. New York: Random House Puzzles & Games, 5th edition, 2003.

[15] J. Kummeneje. *RoboCup as a Measure to Research, Education, and Dissemination*. PhD thesis, Stockholm University and the Royal Institute of Technology, Kista, Sweden, 2003.

[16] S. Buck and M. A. Riedmiller. Learning situation dependent success rates of actions in a robocup scenario. In *PRICAI*, page 809, 2000.

[17] I. Fehérvári and W. Elmenreich. Evolutionary methods in self-organizing system design. In *Proceedings of the 2009 International Conference on Genetic and Evolutionary Methods*, 2009.

[18] W. Elmenreich and G. Klingler. Genetic evolution of a neural network for the autonomous control of a four-wheeled robot. In *Sixth Mexican International Conference on Artificial Intelligence (MICAI'07)*, Aguascalientes, Mexico, November 2007.

[19] FIDE swiss rules. Approved by the General Assembly of 1987. Amended by the 1988 & 1989 General Assemblies.

[20] Wikipedia. Buchholz system — Wikipedia, the free encyclopedia. [Accessed 22-Mar-2009].

[21] C. Gershenson. *Design and Control of Self-organizing Systems.* PhD thesis, Vrije Universiteit Brussel, Brussel, Belgium, 2007.

[22] A. Tyrrell, G. Auer, and C. Bettstetter. Biologically inspired synchronization for wireless networks. *Advances in Biologically Inspired Information Systems: Models, Methods, and Tools*, 69:47–62, 2007.

[23] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Books, 2004.

[24] C. Auer, P. Wüchner, and H. Meer. A method to derive local interaction strategies for improving cooperation in self-organizing systems. In *IWSOS '08: Proceedings of the 3rd International Workshop on Self-Organizing Systems*, pages 170–181, Berlin, Heidelberg, 2008. Springer-Verlag.

[25] X. Yao and Y. Liu. Evolving artificial neural networks through evolutionary programming. In *Proc. of the Fifth Annual Conference on Evolutionary Programming*, pages 257–266. MIT Press, 1996.

[26] L. A. Meeden. An incremental approach to developing intelligent neural network controllers for robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(3):474–485, June 1996.

[27] D. Floreano and F. Mondada. Evolving of homing navigation in a real robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(3):396–407, June 1996.

[28] G. Baldassarre, D. Parisi, and S. Nolfi. Distributed coordination of simulated robots based on self-organization. *Artif. Life*, 12(3):289–311, 2006.

[29] M. Sipper, Y. Azaria, A. Hauptman, and Y. Shichel. Designing an evolutionary strategizing machine for game playing and beyond. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 37(4):583–593, 2007.

[30] A. L. Nelson, E. Grant, and T. C. Henderson. Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, 46(3):135 – 150, 2004.

[31] A. L. V. Coelho and D. Weingaertner. Evolving coordination strategies in simulated robot soccer. In *Proceedings of the International Conference on Autonomous Agents*, pages 147–148. ACM, 2001.