PS Einführung in die Bioinformatik

NC-003-2001 May 31, 2001

Reactive agents, hybrid methods (ANN, GA, ...) in MAS

Report

Johannes Pletzer
yohap@gmx.net

Thomas Fuhrmann
thomas.fuhrmann@aon.at

Academic Supervisor Mag. Dipl.-Ing. Schwaiger Roland

Department of Computer Science
University of Salzburg

Correspondence to:
Universität Salzburg
Institut für Computerwissenschaften und Systemanalyse
Jakob–Haringer–Straße 2
A–5020 Salzburg
Austria

Reactive agents, hybrid methods (ANN, GA, ...) in MAS

Johannes Pletzer and Thomas Fuhrmann
Department of Computer Science
University of Salzburg
A–5020 Salzburg, Austria
yohap@gmx.net, thomas.fuhrmann@aon.at

Abstract

The purpose of this document is to report about the use of reactive agents and hybrid methods (i.e. artificial neural networks (ANN) and genetic algorithms (GA)) in Multi Agent Systems (MAS).
Reactive agents are a special type of agents that do not possess internal models of their environments. They do not plan their actions, they rather act in a stimulus-response manner which is relatively simple. This includes interaction with other agents in basic ways. Despite Reactive agents being quite simple, very complex patterns of behavior emerge considering the agent population globally.
Hybrid methods are the combination of two or more strategies or technologies in order to have a better performance as opposed to using just one single method. For example you can combine GA and ANN to simulate evolution in an MAS. In this example the ANN which controls the wandering behavior of the agents is encoded in a gene and is evolved using a genetic algorithm.

Contents

Chapter 1

Introduction

In this Introduction the basic terms and technologies which we used in our research are described.

1.1 Agents

It is not that easy to define the term "agent" because there are a lot of different researchers dealing with them who have not managed to come up with a definition they agree upon. One definition is that an agent is a piece of software and/or hardware that is capable of acting autonomous in order to accomplish tasks on behalf of its user or creator. Another simple description of an agent is that it is a system that tries to fulfil a set of goals in a complex dynamic environment. For example humans and animals can be seen as very sophisticated agents with many sensors that enables them to survive in this complex dynamic environment called earth. Agents come in many physical guises: Agents can be robots but of course also pieces of software. There are various types of agents for a range of purposes, i.e. internet-search agents, navigation agents, testing agents and help agents, just to name a few.
There are several dimensions to classify agents [Hya96]. Firstly, they can be classified by their mobility. Agents can either be *static or mobile*, which means they are able to move around in real or virtual environments or in computer systems. Secondly, they may be classed as either *deliberative or reactive*. Deliberative agents have a internal reasoning model which enables them to plan their actions. In contrast to that reactive agents act in a relatively simple stimulus-response manner which for example can mean that they are able to sense objects in their environment and then show a specific reaction based on the sensed data. They may also be able to communicate with other agents in basic ways. Thirdly, agents can have various attributes: autonomy, learning and cooperation.
Autonomy means that agents can operate on their own, without explicit human guidance. A key element in the autonomy of agents is their proactiveness, which means the can "take the initiative".
Cooperation is important because obviously this is the reason for thinking about having more than one agent for specific tasks which would be far less useful if they would not be able to communicate with each other.
Learning is another key-feature for truly "smart" agents. You may view agents as "pieces of intelligence" and intelligent beings certainly must have the ability to learn in order to adopt to their environment and increase their performance.
Most agents that have been developed by now have a combination of these three attributes and if it is possible to combine them all in a way that combines their strengths this would be a truly smart agent. But unfortunately this does not seem to exist by now.

1.2 Multi Agent Systems (MAS)

As the term already says, in multi-agent systems a number of agents are put together for a specific purpose. In such an "agent society", agents coordinate their skills, knowledge, goals and plans jointly in order to take actions or to solve problems collectively. These agents may work toward separate but related goals, or toward one single goal. A basic classification for MAS is their homogeneity or heterogeneity.
In *homogeneous MAS* the agents are all exactly or quite the same. The idea is that despite one agent is quite simple and easy to understand, complex patterns of behavior emerge when the agent population is viewed globally. Especially reactive agents are quite suitable for this kind of MAS because they react in simple stimulus-response manner. In homogenous MAS many simple low-level interactions, which are for example agents detecting each other or other objects and simple communications between them, can combine to form highly complex system-level behavior. J. Ferber refers to the behavior of single agents as quantitative and to the behavior of the whole system as qualitative behavior. This effect can be seen in biological systems too. Bees and Ant have very limited cognitive system but have an amazing group-level cognition. An obvious advantage of homogenous MAS is that if some agents behave wrong, which means not goal-oriented, the overall performance of the system does decrease rapidly and not slowly. If however the system would consist of highly specialized interdependent components (as in heterogenous systems), the malfunction of some of them can mean the overall performance suddenly drops to nearly zero.
Heterogenous MAS consist of different types of agents. In such systems each agent or group of agents has a specific specialized task. There are some arguments for this approach [Ben00]: One is that in nature there are many examples for heterogenous system, for example a multi-cellular organism, where each cell has its specific task and these cell together form a complex being. It seems like that teams of specialists perform better than teams of generalists in complex problem solving. Heterogenous systems are more powerful because they are better at breaking a task down into its component parts. A disadvantage of such system is that they are not that easy to handle because every type of agent behaves differently and for example it is more difficult to develop a communication language which all of them understand.

1.3 Artificial Neural Networks

1.3.1 Basics of Neural Networks

Artificial Neural Networks (ANNs) are an abstract simulation of a real nervous system that contains a collection of neuron units communicating with each other via axon connections. Such a model bears a strong resemblance to axons and dendrites in a nervous system.
The first fundamental modelling of neural nets was proposed in 1943 by McCulloch and Pitts in terms of a computational model of "nervous activity". The McCulloch-Pitts neuron is a binary device and each neuron has a fixed threshold logic. This model lead the works of John von Neumann, Marvin Minsky, Frank Rosenblatt, and many others.
Consequently an Artificial Neural Network being an system that is based on the generalization of information processing in biological neural networks there are several characteristics to be considered [Fau94]

- Information processing occurs at many simple elements called neurons
- Signals are passed between neurons over connection links
- Each connection link has an associated weight, that multiplies the signal transmitted

- Each neuron applies an activation function to its net input to determine its output signal.

Basically a neural network is characterized by its

- Architecture - i.e its pattern of connections between the neurons
- Learning Algorithm - i.e its method of determining the weights on the connections
- Activation function which determines its output

In a biological network you can identify neurons as a special type of processing elements with a certain internal state called activation or activity level which is a function of the inputs it has received. Typically a neuron sends its activation as a signal to other neurons. A neuron can send only one activation signal at a time, however this does not exclude the signal being broadcasted to several other neurons.
Because an ANN mimics the biological neural network, it has to resemble the essential parts of a BNN, such as neuron, axons, hillock and more. Currently, to create an ANN, there are two approaches. The first approach is to use experimental chips that simulate neurons and interconnect them to create a network. However, this approach is inefficient due to the expenses and the technologies behind it. The software solution, on the other hand, is much easier because as the network expands, it is harder to upgrade the network through hardware than through software.

1.3.2 Signal transmission in ANNs

As mentioned before Neurons are the basic processing elements of the neural network. In biological neural networks each neuron has several dendrites which connect to other neurons and when a neuron fires (sending electrical impulse), a positive or negative charge is sent to other neurons. When a neuron receives signals from other neurons, spatial and temporal summation occurs where spatial summation converts several weak signals into a large one, and temporal summation converts a series of weak signals from the same source into a large signal. The electrical impulse is then transmitted through axon to terminal buttons to other neurons. The axon hillock plays an important role because if the signal is not strong enough to pass through it, no signal will be transmitted. The terminal buttons are connected to other neurons or muscle cells. The gap between the two neurons is called the synapse. The synapse also determines the "weight" of the signal transmitted. The more often a signal is sent through the synapse, the easier it is for the signal to be sent through.

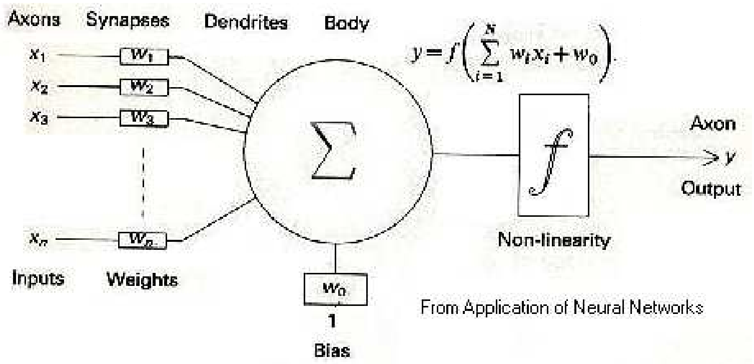

Figure 1.1: Functional Description of a neuron

So to describe a neuron in an ANN you need several functions and variables including weight (a random number generated when the neuron is created), a non-linear function (to determine whether to activate the neuron or not), a method that adds up all the inputs, and a bias/offset value (optional) for the characterization of the neuron.
The non-linear functions that determine whether a neuron fires or not are called activation functions. As with the training methods there exist several functions that generate the output of the neuron with given input signals.
The output of each neuron is the sum of all the inputs multiplying the weights plus the offset value and through a non-linear function. The non-linear function acts like a hillock. Figure 1.2 shows a variety of non-linear functions most frequently used for ANN.

1.3.3 Architecture of ANNs

A typical ANN is organized in layers that are made up of interconnected nodes that contain an activation function. Basically you can divide these layers into three types: input layer, hidden layer and output layer. The input layer is the only layer that receives signals outside the network. The signals are then sent to the hidden layer, which contains interconnected neurons for pattern recognition and relevant information interpretation. Afterwards, the signals are directed to the final layer for outputs. So the actual act of processing is done by the hidden layer via a system of weighted connections. This type of network is called feed-forward network.The term feed-forward expresses that the signal flow is one way. Therefore the signal usually moves straight-forward from the input layer to the output layer. Moreover a feed-forward network does not include feedback loops between neurons.

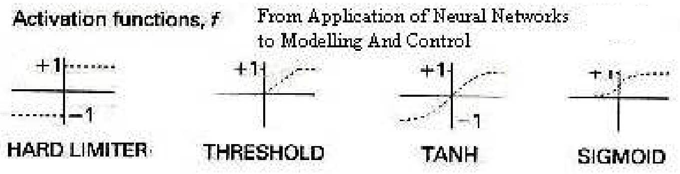

Figure 1.2: Architecture of a typical ANN

A more sophisticated neural network would contain several hidden layers and loops to make the network more efficient and to interpret the data more accurately. However when we talk about ANNs we usually mean feed-forward networks.

1.3.4 Training of ANNs

The process of changing weight could be referred to as the "learning stage." Typically we do not know what the output will look like when presenting the net with an input pattern. The ANN must be trained to classify certain data patterns to certain outputs. This process of training is a series of operations that change the weights on the neural connections, so the user can obtain the output they desire.
For example, the first time the network receives inputs the outputs are some random numbers. The user then has to tell the network what the outputs should be and an algorithm should be applied so the weights are changed to get the outputs wanted.
There are several different types of learning rules. The most commonly used formula for simple networks is called the *Delta Rule*:

$$\Delta w_i = \beta \cdot (t - y_{in}) \cdot x_i$$

Where Δw_i is the change made to the weights, β with $0 < \beta \leq 1$ is the rate for adapting, t is the desired output, y_{in} is the actual output and x_i describes the sum of inputs. The *Delta Rule* is also called the *Perceptron Learning Rule*, and goes back to the early 1960's.
When there are multi-layers in a neural network, the same rule can still be applied to it. The only difference is that the desired output would be the output that is sent to the next layer of neurons and the inputs are from the previous layer of neurons. The network stops training when $(t - y_{in})$ is close or equal to zero because it demonstrates that the network can produce desired outputs. This delta ruled is often used by the most common class of ANNs called *backpropagational neural networks* where backpropagation is an abbreviation for the backwards propagation of error. When a neural network is initially presented with a pattern it makes a random guess what the output may be. Then it makes appropriate adjustments to the connection weights and constantly compares the answer with the actual one until a satisfying output is reached. During this phase we talk about backpropagation. Once the network is trained a satisfactory level of accuracy it may be used for analyzing other data. This no longer comprises any training runs but allows the network to work in forward propagation mode immediately. New input patterns are presented to the ANN and processed by the middle layers as though you would train the network. The output, however, is retained. The output received is the predicted model for the data which can then be used for further analysis and interpretation. This ability of a neural network i.e. to correctly classify input patterns that it has not seen before (not been trained with) is termed generalization
Apart from the Backpropagation training method using the generalized *Delta Rule* there also exist other training algorithms e.g. Kohonen Self-Organizing Maps, Adaptive Resonance Theory (ART), ...

1.4 Artificial Evolution (AE)

The basic idea of artificial evolution is to use the principles of the natural evolution process, the"survival of the fittest" [Dar1859], which created all sorts of life on our planet, for solving complex problems on computers. Obviously such algorithms can be used to create all sorts of "artificial life", which can be either real or virtual robots or agents that for example can learn how to navigate using evolutionary algorithms. Furthermore programs that solve complex problems can be evolved. In comparison to the hand-design of a way to solve a problem the artificial evolution approach allows a gradual emergence [Hus97].

Here is a generic algorithm for evolutionary algorithms (EA) [Ang93]:

```
function Evolutionary-Algorithm(population, size);
begin
   for i from 1 to size do
      fitness[i] := evaluate(population[i]);
   while not(Tester(bestof(population, fitness))) do
   begin
      Select(population, fitness);
      Reproduce(population);
      for i from 1 to size do
         fitness[i] := evaluate(population[i]);
   end;
   return bestof(population, fitness)
end;
```

In this algorithm a population, which can for example consist of programs or agents, with a given number of objects is evolved until one of the objects is “good enough” to stop searching for a better one. The population with which the evolution process is started is generally uniformly distributed over the search space. A defined fitness criterion measures how ”good” or efficient an object is. In every loop the fittest objects are selected and reproduced. The size of the population remains constant because all non-selected objects are replaced by reproduced ones. Then the fitness of the new population, which is also called the new generation, is calculated. If one object in the population fulfils the required level of fitness (this comparison is made by the Tester-function) this object is the result of the EA.
In natural evolution the fitness criterion is generally the ability to adopt to the environment. The species with the ability to survive and to reproduce itself is “selected” for the next generation. But natural evolution never comes to an end and there seems to be no single final solution for the complex problem to live on this earth. A lot of evolution threads coexist and they interfere with the evolution of other species. This mutual interference is called “coevolution”.
There are 3 major variations of evolutionary algorithms:

- Genetic Algorithms (GA)
- Evolution Strategies (ES)
- Genetic Programming (GP)

1.4.1 Genetic Algorithms (GA)

Basically Genetic Algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a very simple chromosome-lime data structure and apply recombination operators to these structures to preserve critical information.
So an implementation of a genetic algorithm usually begins with a population of chromosomes. Then these structures are evaluated and reproductive opportunities are allocated in such a way that those chromosomes which present a better solution to the problem targeted are given more chances to reproduce than chromosomes that provide poorer solutions. The qualitative evaluation of the solutions is usually seen with respect to the population globally.
Strictly interpreted the term genetic algorithm refers to a model introduced and investigated by John Holland (1975) and by students of Holland. However in broader usage of the term a genetic algorithm is any population-based model that uses selection and recombination operators.
Typically, in Genetic Algorithms, bit strings of a fixed length l represent an individual. This does not mean, however, that GA can only solve Boolean problems, for which this coding is directly suitable. More complex data structures can be mapped to bit strings in an appropriate way. However, each additional mapping between the binary and the problem representation weakens the necessary causality between genotype and phenotype. For each optimization problem one has to decide whether to choose a representation tailored for the problem and to adapt the genetic operators, or using a standard GA by coding the decision variables more or less skillfully in a bit string.
A GA population consists of individuals. The selection chooses, at least, two individuals for the next mating with a probability proportional to their fitness. Furthermore, a number c is randomly taken from the set . The descendant receives the first c bits from one parent, the remainder from the other. This recombination process is called crossover (indicating the similarity to recombination of DNA in biological systems) *Crossover* is the main manipulation operator for this evolutionary model. Considering mutation we can also identify two other methods of recombination. Inversion takes a segment of the fixed-length string and removes it from the

object. Then this segment is transcribed back into the original position but in reverse order.*Point mutations* only modify a single position of a string by replacing it with a random value.
Individuals produced in this way replace the parental generation. This procedure is repeated until a termination criterion is reached.

1.4.2 Evolution Strategies (ES)

Evolution-strategic optimization is based on the hypothesis that during the biological evolution the laws of heredity have been developed for fastest genetic adaptation. Evolution-Strategies (ES) imitate, in contrast to the genetic algorithms, the effects of genetic procedures on the phenotype. The presumption for coding the variables in the ES is the realization of a sufficient strong causality (small changes of the cause must create small changes of the effect). So with Evolution strategies the main emphasis is not on developing a structure with a high level of fitness, but for a behavior that meets the requirements of a certain fitness function. So within the evolution process individuals are created that are similar to their parents in behavior but not necessarily in structure. Evolution strategies consider an individual being composed of certain behaviors whereas their interaction is typically unknown. Evolution Strategies used fixed-length real valued strings for encoding. The problem that appears for evolution is how to preserve the behavioral similarity between parent and offspring individuals. This is achieved by use of suitable operators. However with Evolution Strategies there are more types of operators possible than used with genetic algorithms. Intermediate recombination, for instance, allows creation of offsprings by averaging the values of to parent individuals. So we actually do not only remix the components but also mix the individuals' features. However it is not necessary only to manipulate the strings with recombination. Another mutation operator used in ES is to apply Gaussian noise to the parameters. The variance of noise is defined by the following rule(known as the $\frac{1}{5}$ success rule):
The ratio of successful mutations to all mutations should be $\frac{1}{5}$.If it is greater than $\frac{1}{5}$ then increase σ by a constant, if it is less then decrease it [Ang93]

$$\sigma(t) = \begin{cases} \sigma(t-1) \cdot c & \text{if } p > \frac{1}{5} \\ \sigma(t-1)/c & \text{if } p < \frac{1}{5} \\ \sigma(t-1) & \text{if } p = \frac{1}{5} \end{cases}$$

1.4.3 Genetic Programming (GP)

Genetic programming (GP) designates a set of evolutionary processes that generate genotypes representing algorithms. These algorithms are meant to solve a specific problem. The problem solving capability of a genotype is given by its algorithm's capability of approximating a specific input-output relation.
A phenotype is a genotype's representation that is the immediate input for an interpreter or a compiler. The program behavior shown during the phenotype's interpretation or execution of its compiled machine-code representation on the input-output relation gives the problem solving capability of the represented genotype.
So with Genetic programming the individuals of the population or rather their phenotypes can be represented as programs. The genotype-phenotype mapping is an analogy to the biological process of protein synthesis that generates proteins from DNA.
In traditional Genetic Programming GA operators are used for selection and recombination of the individuals. However as opposed to GA, a more expressive programming language such as Lisp is used for describing the genotype. All functions, variables and constants that can occur

are stored as a parse tree. Functions variables and constants that require no arguments become the leaves of the parse tree and are called terminals whereas functions requiring arguments are the branches of the tree.

A simple GP algorithm can be described as follows:

1 Randomly generate a population of N programs made up of functions and terminals

2 Repeat the following until termination condition is satisfied:

   (a) Assign a fitness to each of the programs by executing them on a specified problem and evaluating their performance solving these problems

   (b) Create a new generation of programs by genetic recombination

3 The best program over all generations or the best program at the end of the run is used as the solution produced by the algorithm.

Chapter 2

Related Work

2.1 GA for optimizing task performance in MAS

Arvin Agah and George A. Bekey studied the results of evolution on the task performance of a robot colony using a genetic algorithm [Arv96]. For their robots they used the Tropism System Cognitive Architecture, which is based on the tropisms of the robots, i.e. their likes and dislikes. Each tropism for different entities, which can for example be free spaces, obstacles, food, etc. is encoded by a value. The higher the value the more the robots "likes" the specific entity and therefore it is more probable that it moves towards it. The actions the robot performs is chosen randomly, but actions with higher tropism values are more likely to be performed. These values for a predefined selection of entities and related actions are encoded into a gene and then evolved using genetic algorithms. The fitness function used determines the fitness based on the number of tasks performed by the robot and its energy consumption. Multipliers are used to assign different weights to the tasks and the energy consumption. Robot combination is done by selecting 2 parents from the current generation (better performing ones are more likely to be chosen for recombination) and combining their genes and in order to produce 2 new robots, which are called the offspring. The probabilities for cross-over and mutation determine the outcome of this recombination. If neither happens the offspring is identical to its parents. Agah and Bekey performed an experiment where the robots had to attack predators. After about 40 generations the maximum performance was reached via evolution. They also tried to find out optimal values for mutation probability. It seems like that very low mutation rates (about 0.01) performed better that higher ones in their experiments. They conclude their paper with the statement that evolution with the use of genetic algorithms can indeed be used to evolve a colony of robots with superior task performance, compared with the initial colony.

2.2 ANN and ES in MAS for Navigation in Unknown Environments

Fang Wang and Eric Mckenzie from the University of Edinburgh (UK) developed a multi-agent based evolutionary ANN for general navigation in unknown environments [Wang99]. Their motivation was that real creatures do perform quite well even if they never saw the environment they got into before. Wang and Mckenzie tried to simulate that with virtual creatures. Basically they trained (by evolution) the agents in training environments and then tested how the agent performs in a randomly generated environment it has not trained with. They gave their agents to ability to learn during its lifetime in order to improve the ability of the agents to get along with unknown environments.

The used agents have a simple visual sensor which can sense objects in a range of 90 degrees.

The sensor used cannot differ between different objects, it just senses if the places in vision are occupied by another object, for example a wall or an obstacle, or if it is a place where the agent can move to. The sensor information is encoded to a real value in the interval [0,1] and this is the only input the ANN of the agents gets. The ANN consist of one input for the sensor, 10 hidden neurons and one motion output. There are eleven possibilities of reaction for the agent. It can either move to the left, right or forward, remain stationary or change its viewing direction to 7 other angles. For Evolution Wang and Mckenzie used an (15,100)-Evolutionary Strategy which means out of 100 offspring the 15 parents which performed best survive. They let “nature”, in this case the environment judge the individual agents. In the evaluation function a successful move is rewarded most, a direction change is rewarded less, not moving is neutral and a collision triggers a high penalty. It also encourages the agents to move to places never been before. The function accumulates all moves the agent makes in a one stage of the evolution process which lasts 200 moves. The performance of the agent is evaluated in each stage of evolution by the number of collision that occurred in the 200 steps and the number of spare places access by the virtual creature as a percentage of all free places in the environment. After only 3 learning stages the number of collision dropped to nearly zero and in the next 5 stages the percentage of spare places improved to about 80 percent. After this intensive learning period the agents were confronted with a randomly generated world to test its generalization ability. In this test in the first couple of steps the agents with lifetime-learning do not perform significantly better that those without, but later on those without it begin to tumble over obstacles and produce much more collisions. In their experiments the agents have developed an emergent behavior which is natural, efficient and robust and has not been programmed in advance. One of these behaviors is to move forward in a straight line, which is also a common characteristics of animal movements. Furthermore Fang and Mckenzie showed that lifetime learning is very useful for navigation in unknown environments because it obviously enables the agents to adopt to the new situations.

2.3 Evolving Multiagent Coordination Strategies with Genetic Programming

Thomas Haynes, Sandip Sen, Dale Schoenefeld & Roger Wainwright from The University of Tulsa use a variant of the Genetic Programming Paradigm (GP) for the evolution scheme of their agents [Hay95] . They evolve behavioral strategies for the predator-prey problem that involves multiple predator agents trying to capture a prey agent within a certain world surrounding it. For describing the agents' behavior Haynes, Sen, Schoenefeld and Wainwright rather concentrate on the strongly typed genetic programming paradigm (STGP). In traditional GP all of the terminal and function member sets have to be of the same type . Introducing STGP allows the terminals and function sets describing the behavioral strategies to be of any type although they have to be specified before. Thus the agents' behavioral strategies are encoded in parse trees where the root node is a randomly chosen tack of one of the five choices the agent can make (Here, North, East, West and South). To evolve these strategies presented as programs Haynes, Sen, Schoenefeld and Wainwright have rated and tested them by generating random pursuit scenarios where each program is run for 100 time steps. For each time step the fitness of the program is determined by use of a special evaluation function with arguments such as Distance of predator from prey, Number of Moves, Capture situation after end of simulation .... The main task of the STGP algorithm is now to evolve the program that is used for a predator to choose its moves. In these test scenarios the prey to be captured either moves randomly or uses a *move away from the nearest predator* algorithm. As a result of their work they found out that strongly typed genetic programming is able to generate effective strategies for predator agents capturing a randomly moving prey or a prey that moves away from the nearest predator. They compared their experimental results with previous works from Richard Korf [Kor92] where deterministic algorithms were used. Using their STGP programs preys were usually more often captured than with deterministic methods like the *max norm* and the *Manhattan distance* algorithms used by Korf. However all the STGP programs tested poorly results when using special strategies for the prey's movement: a prey moving in a straight line, a prey that maximizes its distance from the current positions of the predators and most surprisingly a prey that does not move at all. However a basic problem encountered in the work of Haynes et al. is the lack of any explicit communication between the predator agents for coordinating their actions an developing a cooperative behavior. Too often deadlock situations occur where one agent obstructs one other e.g. if trying to move to the same position at one time step.

Chapter 3

Evolving local agents for global classification problems

In this chapter we try to describe possible ways for designing agents that become local experts. So what is a local expert? It is an agent that has 2 basic patterns of behavior: *exploration* and *localization*. On one hand it tries to explore as much of its environment as it can, for example by moving in a straight line as long as possible and moving away from its starting point as far as it can. On the other hand it has the "desire" to become a local expert which means it tries to find out what is around it and therefore does not move away too far from its starting point. The starting point may be set randomly but it might also be useful to give the agent the ability of changing the area it is initially set to another where for example no other agent is. The motivation of designing such agents are that they might be very efficient when it comes to classification tasks, for example classify a map in certain regions (streets, woods, mountains,...). For such tasks it would be quite efficient if the agents move along lines or "borders" which separate different regions from another. They should at first try to find a border and then move along it until it is no longer possible or until they get back to the point they first encountered the border. Then they should verify if the region they found is actually a homogenous region or if other regions are inside of it. For example if a agents discovers that his regions is a forrest this does not mean that inside this forrest is no pond or a mountain. If an agent has finished the classification it could move to another area and try to classify it. Of course it depends on the environment to explore when the agent can be "satisfied" with its results, the question is how exact his classification should be verified.

3.1 Creating local experts with an evolved ANN

One of our ideas for creating local expert agents is to "let evolution do the task". It is a try and we'll describe how it might work according to our knowledge. The basic idea is to use an GA-encoded ANN as a neuro-controller for steering the agents. The agent has a visual sensor to sense objects and the type of area it is in. The critical part of course is to find a fitness-function that leads the evolutionary process in the right direction. One part of the function must be to reward agents that stay in a homogenous area and move along a border. They also should be encouraged to explore, but only or preferable in the area they are initially set to. If they sense the beginning of a different area they should either turn left or right and then walk along the border in order to find the whole border of the area. Therefore the fitness function must also include a reward for finding the border of the whole area, which is accomplished if the agent "walks around" the area. The behavior described so far is only a basic type of a local agent. As described above more sophisticated ones would explore the inside of the area too in order to verify their assumption that it really is one homogenous area they found. But if the algorithm with which we try to generate such agents here does not even produces this type of basic agents it most certainly won't produce more sophisticated ones.

3.2 Creating local expert agents with evolved tropism values

Inspired by [Arv96] another way of creating such agents is to use tropism values. It might be useful to have one value for staying in the same area, one for entering another area and maybe one for moving along separation lines. These values could easily be encoded in a gene and then be evolved using the fitness function described above. Probably the agents would prefer staying in the area there were initially set and walk along separation lines rather than enter another area.

3.3 Further ideas and issues

Until now we just focussed on one agent, but if more than one agent has to classify an environment it is of course interesting to take a look on how these agents could interact. For example the following question is raised: If an agent encounters an agent is his area, should they united to classify the region together or should one of the agents move to another area and classify it. The decision to make should depend on the size of the area, but probably the agents do not know about its size already. It is also not that easy to find a simple and efficient way of communication between the agents. It would be quite useful for them to talk about information they already found about the environment to classify.

3.4 Creating agents with explorative and conservative behavior with an evolved ANN

Our first approach for designing suitable agents for global classification problems is to let an ANN do the task of controlling the agents' explorative behavior. The main problem, however, is to find a good balance between this explorative "feature" and the desired requirement of limiting the agent to a local region.

3.4.1 Structure of the ANN controlling the Agents' movements

Due to the fact that choosing an appropriate ANN architecture for a specific problems is often based on experimental results and experience, we take a quite simple design for controlling the Agents' movements making it easier to evolve and to overview its evolutionary process.
For the ANN's input layer we use two neurons, one providing a value for an angle φ that describes the relative position regarding the agent's initial starting point and a second that covers the distance D. This means our agent has a "feeling" for distance and can tell where it is in relation to its starting point. These values are processed by one hidden layer consisting of six neurons that are initially connected with all other neurons (i.e. with all input and output neurons). The output layer is made up of four neurons that define the directions the agent is possible to move i.e. forward, backward, left, right.
In this case the translation of the neurons values into an action is based on the "winner takes it all" principle thus meaning that only the action connected with the most powerful value is carried out.
As with every ANN the main values for being adapted or rather evolved are the threshold values for a neuron's to fire and the weights of the connection links. For evolving the net by a Genetic algorithm we also consider the actual connections of the hidden layer for being changed.

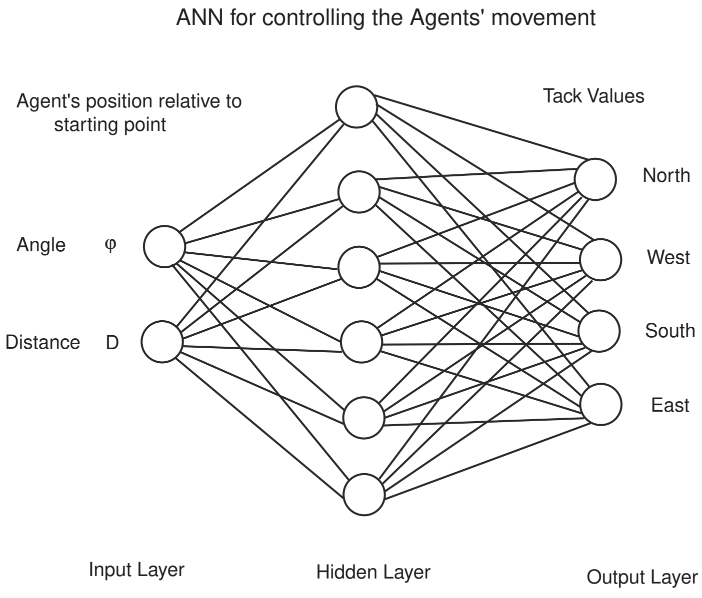

Figure 3.1: Architecture of the evolvable ANN

3.4.2 Fitness function for evolution of agents

In order to reward the desired behavior which is explorative but also conservative behavior we use the following fitness function. The function consist of two parameters:

- C_d which is the maximum distance from the starting point to all of the places an agent visited.
- C_p which is how many different places an agent has explored as percentage of how many steps the agent has performed.

C_d is a measure of how explorative or conservative an agent is. A very high value means that it moved far away from its starting point whereas a low value indicates the the agent never moved far away from its initial position. C_p is obviously needed because we want an agent that explorer as much as possible. In order to balance to two values we normalize them to a value in the interval [0,1], where a value of 1 indicates maximum fitness. C_p is already in the right interval because it is a percentage value, i.e. calculated by dividing the actual number of visited places by the number of steps the agent has moved. For normalizing C_d it is at first necessary to determine the minimum distance an agent has to move away from its initial point in order to be able to reach all the different places it visited. If a agent has a certain C_d it is not possible to reach more than $\pi \cdot (C_d)^2$ places. This leads us to a minimum value of $\sqrt[2]{\frac{C_p}{\pi}}$ for C_d. When dividing this minimum value by the actual value of C_d we get our desired normalization for C_d, a value in the interval [0,1] where 1 means maximum fitness is reached. Now it is necessary to combine C_d and C_p in a fitness function f:

$$f = w_d \cdot C_d + w_p \cdot C_p$$

Adding the 2 weights w_d and w_p to the values gives us the possibility to control the evolution process by specifying whether to put the emphasis on maximizing the visited places or the localization behavior of the agent. By defining that $w_d + w_p = 1$ we get a value in the interval [0,1] for the fitness of an agent.

3.4.3 Possible results

If the two fitness parameters are well balanced the outcome of the evolutionary process should be an agent that does not move far away from its initial starting point but explores as many places as possible. Probably it will move in a circular region in a certain movement pattern which lets the agent move in a way that most of the circular area is explored.

3.5 Agents with tropism values

The basic idea of creating an agent with explorative and conservative behavior is to use one tropism value for moving away from the starting point and another for moving back to it. These values do not remain constant in the lifetime of the agents but are changed dynamically based on the current distance from the starting position. If the agent is very near to his initial point the tropism value for moving away is very high and this action is very likely to be chosen by the agent, but as it moves away from the point the tropism value decreases and at the same time the value for moving back to the initial point increases. So there comes a point when the agent turns from over from explorative to conservative behavior. Of course it is not wise to let the agent move back to where he came from because all places on this way were already visited earlier. It should move into another direction to find places that it has not been before but are not that far way from its starting point.

Appendix A

Protocols

A.1 23.03.2001

1. Person responsible for protocol: Johannes Pletzer
2. Participating persons: Roland Schwaiger, Johannes Pletzer, Thomas Fuhrmann
3. Topics talked about:
   - Paper should be "compact", which means it should be readable for persons without specific knowledge on the topic
   - We should put the emphasis on "artificial life" rather than focusing on the formal aspect of the topic
4. New assignments:
   - Write two section for the introduction: Agents and multi-agent systems (MAS) and artificial neural networks (ANN)
   - Reading some articles in order to be able to discuss the topic

A.2 30.03.2001

1. Person responsible for protocol: Johannes Pletzer
2. Participating persons: Roland Schwaiger, Johannes Pletzer, Thomas Fuhrmann
3. Topics talked about:
   - Results of previous assignment was discussed and approved with minor changes
   - Maybe focus our research on lifetime learning in addition to generation-based learning in an evolving MAS
4. New assignments:
   - Improve image quality, maybe some additional image of ANN-layers
   - Write another section for the introduction: Artificial evolution (AE)
   - Further search on articles dealing with evolution of agent behavior

A.3 6.04.2001

1. Person responsible for protocol: Johannes Pletzer
2. Participating persons: Roland Schwaiger, Johannes Pletzer, Thomas Fuhrmann
3. Topics talked about:
   - Results of previous assignment was discussed and approved with minor changes
4. New assignments:
   - Write a new chapter "related work" with descriptions of the work of other peoples. 3 sections for examples for genetic programming, genetic algorithms and evolutionary strategies.

A.4 27.04.2001

1. Person responsible for protocol: Johannes Pletzer
2. Participating persons: Roland Schwaiger, Johannes Pletzer, Thomas Fuhrmann
3. Topics talked about:
   - Results of previous assignment was discussed and approved with minor changes
4. New assignments:
   - Finish chapter "related work" with an example for GP
   - Search for related papers and/or find strategies for creating agents that become local experts for a certain area.

A.5 10.05.2001

1. Person responsible for protocol: Johannes Pletzer
2. Participating persons: Roland Schwaiger, Johannes Pletzer, Thomas Fuhrmann
3. Topics talked about:
   - Review of the whole paper, correction of some minor mistakes.
   - Results of previous assignment were discussed and problems were found in the suggestions on how to create local experts.
4. New assignments:
   - Find suggestions on creation of agent that has explorative and conservative behavior.

A.6 17.05.2001

1. Person responsible for protocol: Johannes Pletzer
2. Participating persons: Roland Schwaiger, Johannes Pletzer
3. Topics talked about:
   - Suggestion for creating local experts with evolved behavior was discussed.
4. New assignments:
   - Write a more detailed description of how to evolve local expert agents, especially about ANN structure and fitness function.

Bibliography

[Fau94] L. Fausett, Fundamentals Neural Networks, New Jersey, Prentice-Hall 1994

[Hya96] Hyacinth S. Nwana, Software Agents: An Overview, Ipswich, UK, BT Laboratories 1996

[Ben00] Benjamin McGee Good, Evolving Multi-Agent Systems: Comparing Existing Approaches and Suggesting New Directions, 2000

[Ang93] Peter John Angeline, Evolutionary Algorithms and Emergent Intelligence, Ohio, USA, 1993

[Dar1859] Charles Darwin, On the Origin of Species, 1859

[Hus97] P. Husbands, I. Harvey, D. Cliff and G. Miller, Artificial Evolution: A New Path for Artificial Intelligence?, University of Sussex, Brighton, UK, 1997

[Rech94] Ingo Rechenberg, Evolutionsstrategie '94, Frommann-Holzboog, Stuttgart, 1994

[Wang99] Fang Wang and Eric Mckenzie, A Multi-agent based Evolutionary Artificial Neural Network for General Navigation in Unknown Environments, Division of Informatics, University of Edinburgh, UK, 1999

[Arv96] Arvin Agah and George A. Bekey, A Genetic Algorithm-Based Controller for Decentralized Multi-Agent Robotic Systems, Japan/USA, 1996

[Hay95] T.Haynes, S. Sen, D. Schoenefeld and R. Wainwright, Evolving Multiagent Coordination Strategies with Genetic Programming, Department of Mathematical & Computer Sciences, The University of Tulsa, 1995

[Kor92] Richard E. Korf A simple solution to pursuit games, In Working Papers of the 11th International Workshop on Distributed Artificial Intelligence, pages 183-194, February 1992