Comparison of two methods for evolving recurrent artificial neural networks for robot control

(HS-IDA-EA-98-110)

Lúdvík Gudjónsson (b95ludgu@ida.his.se)

Institutionen för datavetenskap
Högskolan Skövde, Box 408
S-54128 Skövde, SWEDEN

Examensarbete på det datavetenskapliga programmet under vårterminen 1998.

Handledare: Zoltán Biró

Comparison of two methods for evolving recurrent artificial neural networks for robot control

Submitted by Lúdvík Gudjónsson to Högskolan Skövde as a dissertation for the degree of BSc, in the Deptartment of Computer Science.

980612.

I certify that all material in this dissertation which is not my own work has been identified and that no material is included for which a degree has previously been conferred on me.

Signed: ______________________________________

Key words: recurrent artificial neural networks, genetic algorithms, robot controllers, evolution, hunter-prey problem, evolutionary robotics

Abstract

In this dissertation a comparison of two evolutionary methods for evolving ANNs for robot control is made. The methods compared are SANE with enforced sub-population and delta-coding, and marker-based encoding. In an attempt to speed up evolution, marker-based encoding is extended with delta-coding. The task selected for comparison is the hunter-prey task. This task requires the robot controller to posess some form of memory as the prey can move out of sensor range. Incremental evolution is used to evolve the complex behaviour that is required to successfully handle this task. The comparison is based on computational power needed for evolution, and complexity, robustness, and generalisation of the resulting ANNs. The results show that marker-based encoding is the most efficient method tested and does not need delta-coding to increase the speed of evolution process. Additionally the results indicate that delta-coding does not increase the speed of evolution with marker-based encoding.

Sammanfattning

Denna dissertation behandlar en jämförelse mellan två metoder för att evolvera artificiella neurala nätverk för en robotkontroller. De två metoderna är "marker-based encoding" samt "SANE with enforced sub-populations and delta-coding". "Marker-based encoding" har i detta arbete utökats med "delta-coding" i ett försök att minska antalet generationer för att hitta en lösning på problemet. Problemet som valts för att jämföra de båda metoderna är "the predator-prey task". Problemet kräver att robotkontrollern använder sig av minne från föregående steg eftersom bytet (eng. prey) kan lämna robotens sensortäckningsområde. För att evolvera en lämplig strategi för att roboten ska hantera denna komplicerade uppgift har "incremental evolution" använts. Jämförelsen är baserad på den CPU-tid som evolutionen kräver, samt komplexitet (antal noder och kopplingar), brustolerans och generaliseringsförmåga för evolverade ANN.

Den föreslagna hypotesen är att "marker-based encoding" är minst lika bra på att hantera detta problem som "SANE with enforced sub-populations and delta-coding" och att CPU-tiden kan reduceras om "delta-coding" används.

Resultaten visar att små förändringar i problemet kan kräva stora ändringar i ANN-topologin. Eftersom faktorerna som påverkar hur många noder som skall väljas för ett specifikt problem är okända är det en stor nackdel att antalet dolda noder måste bestämmas i förväg i "SANE with enforced sub-populations and delta-coding". Resultaten bekräftar hypotesen att "marker-based encoding" hanterar problemet bättre och behöver inte "delta-coding" för att hitta en lämplig lösning. Samtidigt motsäger resultaten hypotesen om att "delta-coding" snabbar upp evolutionsprocessen.

Table of contents

1 Introduction

As humans are not very good at designing complex systems. The until now best known trick is the divide and conquer method, where a complex task is decomposed into a number of simpler tasks. The *divide and conquer* method has not proven efficient in robot controller design (Harvey *et al.*, 1996a). The two main reasons for these difficulties are:

It is extremely difficult to coordinate the different subparts of a robots, both at the level of mechanics and of the control system (Nolfi *et al.*, 1994), and as the system complexity grows the number of possible interactions between subsystems grows exponentially (Harvey *et al.*, 1996a).

As robots interact with an external environment each motor action has two different effects: it influences how well the system performs on a given task and gives rise to the next input stimuli, i.e. motor actions may have long-term consequences. For these reasons determining the correct motor action can be extremely difficult (Nolfi *et al.*, 1994).

1.1 Problem domain

In this project evolutionary methods are used to evolve the internal dynamics of robot controllers.

The robot is placed in a dynamic world and is to act as a predator hunting a prey. The dynamics of the world is the prey that is trying to evade the robot using a prey movement algorithm. The environment is a grid world in which the robot controller at each time step has to decide a direction in which to move. The prey can move out of the robot's sensory range and therefore a successful handling of the problem includes evolution of some short-term memory to decide in which direction to move, based on the prey's position in the previous time step (when the prey is out of sensor range).

The complex general behaviour required to perform well in this problem domain is very difficult to evolve. Gomez&Miikkulainen (1997) suggested incremental evolution to create controllers that follow intelligent strategies in stead of a "mechanical" behaviour like moving around in circles hoping to bump into the prey.

The reason for conducting the experiments in a simulated environment is that the random behaviour of the robot exhibited in the beginning of the evolution can create controllers that when incorporated in physical robot may be hazardous to both its robot and their environment. This is also a problem when using reinforcement learning, as reward or punishment is only given at the end of a run. Another reason is that artificial evolution requires that each member of every generation must be evaluated, which would take far too long time to perform using a physical robot.

1.2 Purpose

The purpose of this project is to compare robot controllers for autonomous mobile robots evolved using two different evolution methods one described in Gomez&Miikkulainen (1997) and Fullmer&Mikkulainen (1991) and the other in Moriarty&Miikkulainen (1995). The robot controllers are to fulfill their tasks of catching an "evading" object in a dynamic world, i.e. keep in mind where an object was last time it was detected and use that information in its decision policy. One difficulty associated with this problem is that information on the quality of each decision is not available, which implies that performance can only be measured after a sequence of decisions has been made. The dynamic world also requires generalized solutions and robustness in the solutions, as the number of possible states is very large.

A possible real world problem that this domain can be related to is the design of robot controllers for robots that are interacting with a dynamic environment where people and possibly other robots are entering and disappearing from its sensory range. The decision policy must then use memory to decrease the chance of selecting a heading that would most likely lead to collision.

2 Background

In this chapter artificial neural networks (ANNs), genetic algorithms (GAs) and previous work on evolving ANNs will be presented.

2.1 Artificial neural networks (ANNs)

ANN is a network of simple processors that are abstractions of biological neurons found in the brain. ANNs use a distributed representation of the information learned in the network resulting in robust and fault-tolerant behaviour, which may for example be used for pattern recognition tasks and in robot controllers. To develop the preferred behaviour training is most commonly used instead of programming (Taylor, 1995).

2.1.1 Brief history

In 1943 McCulloch and Pitts introduced the fundamental idea of neural activity via thresholds and weighted sums (Russell&Norvig, 1996; Hecht-Nielsen, 1990). In 1957 and 1958 the first successful neurocomputer, the Mark I, was developed by Frank Rosenblatt, Charles Wightman and others (Hecht-Nielsen, 1990). Before 1970 most work in neural networks focused on one layer networks (also known as perceptrons), but in 1969 Minsky and Papert (Russell&Norvig, 1996) published a paper where they criticised the unguided experimentation and lack of mathematical rigor that characterised much of the early work. They also stabilised the linear separability criterion for task that could be accomplished with perceptrons, which effectively means that ANN with no hidden units can only approximate the simplest functions. This criterion caused a decrease of interest in neural networks until 1980s when the problem of training multi-layered networks was addressed. It was also shown that single hidden layer is enough to represent any continuous function and two hidden layers are enough to represent any function (Russell&Norvig, 1996).

2.1.2 Why artificial neural networks?

> "Problems in industry and business are all too frequently beyond the scope of our present generation of computers. They run into trouble if data is incomplete or contains errors, if a "best guess" is needed, or if it isn't clear how a problem should be tackled." (Taylor, 1995, p 1)

This is also the case in robot controllers where the sensory inputs can contain noise and it is very difficult to give a set of rules that can handle all possible situations.

As described by Taylor (1995) some advantages of ANNs are that they can be trained to classify poorly structured inputs, they are robust against noise in training data (and internal noise), and loss of neurons or connections, and have the ability to generalize to novel data from a set of examples.

2.1.3 Network building blocks

ANNs are abstractions of the structure and the functions of biological neural networks, such as the brain. Figure 2.1 shows the different parts of a biological nerve cell or neuron.

The neuron sums the signals from other neurons received via synapses. When the sum of signals reaches a certain threshold, the neuron fires transmitting an electronic signal to the neurons connected to its axonal arborization (the other end of the axon). The synapses transmit the signal to the body of the other cells. The synapses can effect the strength of the signal and can be excitatory or inhibitory, contributing positively or negatively to the sum of the activation on the receiving neuron. The ability to change the effects of synapses is called Hebbian learning (Hecht-Nielsen, 1990), and is thought to form the basis for learning (Russell&Norvig, 1996).

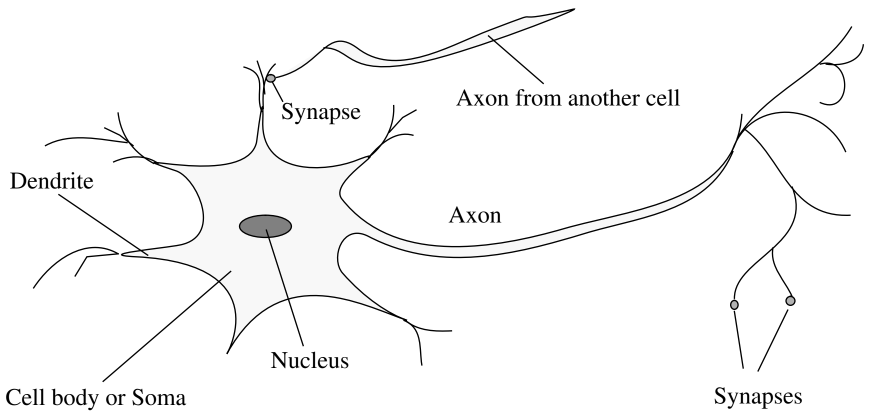


Figure 1: The parts of a nerve neuron. From Russell&Norvig (1996)

In the context of ANNs, the terms *nodes* or *units* are used for the processes inspired by neurons. Synapses are refereed to as *links*. Each link has a numeric weight value associated with it.

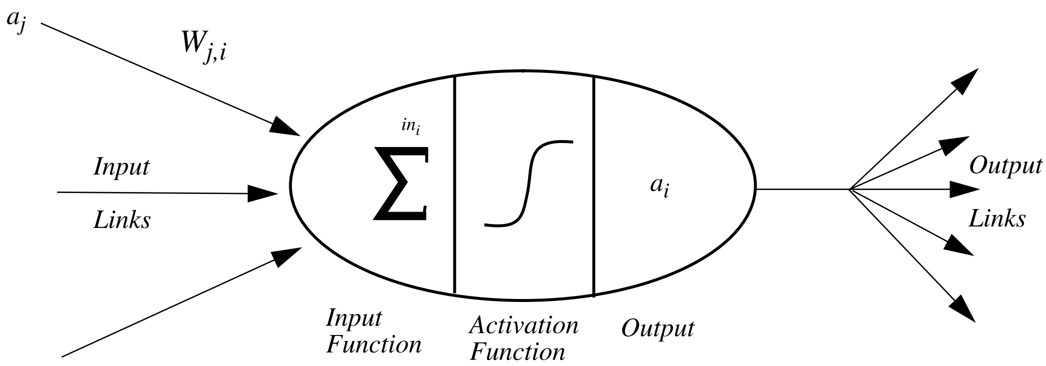
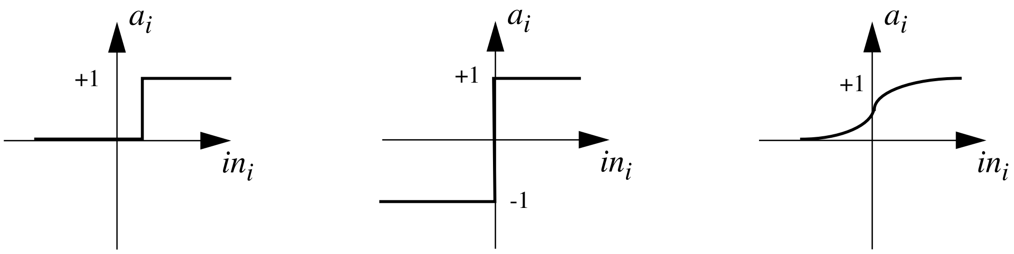


Figure 2: A unit. From Russell&Norvig (1996)

The computation inside a unit is split into two components: a linear component called input function computes the sum of the input values multiplied with their respective weights, and a (usually nonlinear) component called activation function transforms the weighted sum into the units activation value. In figure 2. this is exemplified, there a_j is the output of node j, $W_{j,i}$ is the weight value of the synapse from node j to node i, in_i is the sum of inputs from synapses, g is the activation function, and a_i is the activation value. Three common activation functions are shown in figure 3.



(a) Step function (b) Sign function (c) Sigmoid function

Figure 3: Three different activation functions. From Russell&Norvig (1996)

2.1.4 Network structures

According to Russell and Norvig (1996) there are a variety of kinds of network structures, each possessing different computational properties. The main distinction is between feed-forward and recurrent networks.

In feed-forward networks the links are unidirectional and the net contains no cycles. Hence the activation from previous time steps play no role in computation as there are no feedback connections. Therefore a feed-forward network can only implement reactive agents, i.e. the agent can not use the information of previous inputs or actions as basis for behaviour decisions.

In fully-recurrent networks links can form arbitrary topologies. Hence the network can have short-term memory and activation in previous time steps can be considered in computation. This enables more complex agents to be implemented. On the other hand certain types of recurrent networks can become unstable, oscillate, or start exhibiting chaotic behaviour. It can take long time to compute a stable output and designing a learning algorithm is a problem.

Elman (1990) has described an implementation of a partially-recurrent ANN that has the ability to use the activations of previous time steps in computation, by copying the state of the hidden units in previous computations to extra input units called context units.

Kolmogorov proved that the class of multilayer networks as a whole can represent (or approximate) any desired function (Hecht-Nielsen, 1990), but any specific network can have too few hidden units (Russell&Norvig, 1996). If on the other hand the network has too many hidden units it can memorize the training data in the form of a look up-table, which gives poor generalization and therefore bad result when encountering examples that are not in the training set. “Designing a good topology is, however, a black art.” (Russell&Norvig, 1995, p 583)

2.1.5 Training ANNs

The knowledge of the ANN is in the weights, therefore the representation of the knowledge is distributed. The networks weights are trained using some update rule. As described by Taylor (1995) there are three classes of learning:

- Supervised: Training data is provided, which tells the desired output for every given input. Weights are then adjusted to minimize the difference between the desired and actual outputs for each input pattern ("Learning with a teacher").
- Reinforcement: The network receives a global reward or penalty signal. Weights are changed to develop behaviour maximizing the probability of reward and minimizing probability of penalty ("Learning with a critic").
- Unsupervised: The network is able to discover statistical regularities in the input space and by itself develops different modes of behaviour to represent different classes of inputs.

The most popular leaning algorithm (i.e. algorithm for adjusting weights in ANN) is called back-propagation (BP).

> "Back-propagation provides a way of dividing the calculation of gradient among the units, so the change in each weight is attached, using only local information." (Russell&Norvig, 1995, p 582)

BP is a supervised learning update rule that is actually performing gradient descent in the weight space, therefore it is not guaranteed that a global optimum will be found as the process can get stuck in a local optimum. This has turned many researchers to other methods for searching through the weight space, one of the promising methods is using genetic algorithms to find a global optima in the weight space.

2.2 Genetic algorithms (GAs)

For the first pioneers of computer science the main motivation was the creation of artificial life and artificial intelligence. They were motivated by the visions of the imbuing computer programs with intelligence, with the lifelike ability to self-replicate and with adaptive capability to learn to control their environments (Mitchell, 1996).

2.2.1 Brief history

Computer scientists in the 1950s and 1960s studied evolutionary systems based on the idea that artificial evolution could be used for optimization in engineering problems (Mitchell, 1996). The general idea behind this work was to evolve a population of candidate solutions to a given problem using operators inspired by natural evolution. The field of evolutionary computation has since developed in three main streams: evolution strategies (ESs), evolutionary programming (EP), and genetic algorithms (GAs). Of these three strategies, this dissertation focuses only on GAs.

GAs were invented in the 1960s by John Holland and developed by him and his students and colleagues in the 1960s and 1970s at the University of Michigan (Mitchell, 1996). Holland's aim was not to design algorithms to solve specific problems but to study adaptation as it occurs in nature and to develop ways to import the mechanisms of natural adaptation into computer systems. The method that Holland used for moving one population of "chromosomes" from one generation to another is by means of natural selection, crossover, mutation, and inversion. Each chromosome consisted of "genes" (bits). The chromosomes that were to reproduce were selected and the fitter ones were to produce more offsprings on average. To simulate biological recombination, crossover mechanisms exchanged subparts of two parent chromosomes. Random mutation then changed the value of some genes in the child chromosome.

Researches today often use the term "genetic algorithm" to describe something very different from Holland's original conception (Mitchell, 1996), however with the exception of inversion, the building blocks used by Holland are still there in some form.

2.2.2 Why evolution?

Evolution can be seen as a massively parallel search through a huge space of possible solutions to a problem, where the "problem" is to create an organism that can survive and flourish in a given environment where the "solutions" are the genetic blueprints for different organisms. The rules of evolution are simple to formulate but their effects are hard to predict and applying them repeatedly can result in very complex structures (Mitchell, 1992).

2.2.3 Why genetic algorithms?

According to Mitchell (1996), Holland's arguments were as follows: GAs work by discovering, emphasizing, and recombining good "building blocks" of solutions in a highly parallel fashion. The idea is that good solutions tend to be made up of good building blocks.

> "Denker's remark that "neural networks are the second best way of doing anything" has been extended with "and genetic algorithms are the third"" (Russell&Norvig, 1995, p 621)

When there is no known algorithmic method to solve the problem and the problem can not be solved using an artificial neural network, a promising method is to use genetic algorithm.

2.2.4 Some terminology

The terminology used in the context of GAs originated from biology, but in GAs the terms are much simpler than the original biological terms. Here the terminology used is the same as described by Mitchell (1996).

Chromosomes are strings of DNA that serve as a blueprint for the organism. The chromosome can be divided into *genes* that can roughly be thought of as the encoding of a trait. The different settings for a gene are called *alleles*. Each gene is placed at a particular *locus* (position) in the chromosome. More complex organisms have multiple chromosomes in each cell. The complete set of chromosomes is called *genome* and the term *genotype* refers to a particular set of genes in a genome. Individuals that have the same genomes are said to have the same genotype. Finally the biological organism's *phenotype* is its physical and mental characteristics.

2.2.5 The elements of genetic algorithms

In GAs the term chromosome refers to a possible solution to a problem. It is often encoded as a bit string. Genes are either single bits or blocks of bits that are particular elements of a candidate solution. A bit string can also be an implementation of an integer or a real number. Sometimes the gene is coded directly as an integer or real number.

The GA most often uses a fitness function to assign fitness to each chromosome in the current population depending on how well the chromosome solves the problem. Three operators are then applied to produce the next generation.

- Selection: Selects chromosomes in the population for reproduction. The fittest individual's chromosome is most likely to be selected often for reproduction. This operator simulates the selection occurring in nature.
- Crossover: Randomly selects a locus and exchanges the genes sequences before and after that locus to create two offsprings. For example the genes formed by the strings 1101110110 and 0100111111 could be crossed over after the forth locus to form the two offspring 1101111111 and 0100110110. Two point crossover is a variation of the same but then there are selected two random locus and they used to produce two offspring. For example the same genes shown in previously could be crossed using the third and sixth locus resulting in the following offspring 1100110110 and 0101111111. This operator simulates the recombination of DNA strings from two organisms.
- Mutation: Randomly flips some bits in the chromosome, for example the chromosome 1101110110 could be mutated in the sixth position resulting in the following chromosome 1101100110. This operator simulates the DNA copying errors occurring in reproduction in nature.

2.2.6 Example of a simple genetic algorithm

Genetic algorithms including those used in this dissertation will be based on the same principles as the simple GA described by Mitchell (1996). The following is a description of this basic algorithm:

1. Start with a randomly generated population of n chromosomes.
2. Calculate the fitness of each chromosome in the population.
3. Repeat the following steps until n offsprings have been created:
   a. Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can become a parent more than once.
   b. Given a crossover probability, cross over the pair at a randomly chosen point to form two offsprings. If no crossover takes place, form two offsprings that are exact copies of the parents.
   c. Mutate the offspring at each locus with the mutation probability, and place the resulting chromosome in the new population.

4. Replace the current population with the new population.

5. Go to step 2.

2.3 Agents

A variety of definitions of agents has been offered by researchers (Franklin&Graesser, 1996, for an overview). One definition is the AIMA agent (Franklin&Graesser, 1996) described in Russell&Norvig (1995) as follows:

> "An **agent** is anything that can be viewed as **perceiving** its environment through **sensors** and **acting** upon that environment through **effectors**." (Russell&Norvig, 1995, p 31, original emphasis)

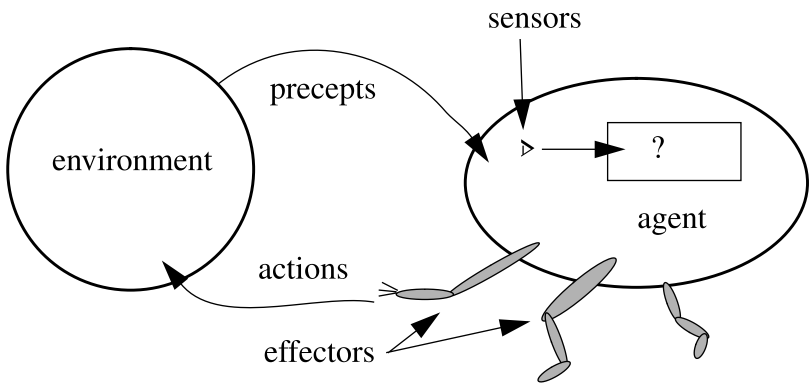


Figure 4: Agent. From Russell&Norvig (1996)

Autonomous was first used in the Greek city states whose citizens made their own laws, as opposed to living according to laws of an external power (Steels, 1995). The term is derived from: *autos* (meaning self) and *nomos* (meaning rule or law), and means essentially: "be relatively independent of" (Steels, 1995). This term has often been used in the meaning "not under the immediate control of human" (Russell&Norvig, 1996). Ziemke (1997) advocates a notion of autonomy that is:

> "... not 'independence of environment' (a 'freedom' most artefacts have), but rather an agent's freedom to form, adapt and utilize the mechanisms that allow it to actively embed itself in its environment, and the capacity to flexibly use this embedding to guide, rater than determine, its own behaviour." (Ziemke, 1997, p 9)

He continues concluding that

> "... autonomy in its encompassing sense should be viewed not as a mere technical feature, but as a key issue in the study of intelligent behaviour in general, and that of robot learning in particular, namely the question how agents/species in the course of evolutionary and individual development enact their *own* 'worlds' and develop modes of interaction with them." (Ziemke, 1997, p 9, original emphasis)

In this dissertation the agents evolved are to be autonomous. Beer (1995) defines an autonomous agent as:

> "... any embodied system designed to satisfy internal or external goals by its own actions while in continuous long-term interaction with the environment in which it is situated." (Beer, 1995, p 1)

The agents in this dissertation are situated in a simulated environment but the ultimate goal of any robot controller is to become part of an embodied system in a real environment.

2.4 Evolution of artificial neural networks

A major issue in the field of ANNs is the design of the architectures. There are strong biological and engineering evidence supporting that the information processing capability of a neural network is determined by its architecture (Liu&Yao, 1996), however, the choice of problem-specific architecture still remains a poorly understood task (Braun&Weisbrod, 1993). If the ANN is too small then it will not be capable of forming a good model of the problem. If on the other hand the network is too big then the ANN may overfit becoming a lookup table of the training data and therefore have very poor generalization ability (Liu&Yao, 1996). There is no systematic method to design problem specific architectures. Attempts have been made to automate this design process using constructive or destructive algorithms. Constructive algorithms start with the smallest possible network and gradually increase the size of the network until the performance starts levelling off while a destructive algorithm does the opposite: it starts with a large network and with unnecessary layers, nodes and connections are then deleted during training.

> "The problem of designing optimal architecture for an ANN can be formulated as a search problem in the architecture space and where each point represents an architecture. Given some performance criteria about architectures, the performance level of all architectures forms a surface in the space. Then optimal architecture design is equivalent to the highest point on this surface." (Liu&Yao, 1996, p 1)

For these reasons many researchers have in recent years turned towards the use of evolutionary methods for evolving the architectures of ANNs.

Evolutionary algorithms offer a flexible alternative to back-propagation and other hill-climbing search methods, since they are also applicable to a much broader range of architectures and do not require examples of correct behaviour (Moriarty, 1997). As examples of correct behaviour are not needed for every step, evolutionary methods are good for reinforcement learning, where traditional hill-climbing algorithms are not efficient. If the architecture of ANN is being evolved, a fitness for each possible solution in all generations of the process must be calculated. This implies that every individual must be trained which is very time consuming and sensitive to the initial conditions of the training. Also if the measure of the fitness is the root mean square (RMS) sum of errors on the training set then the tendency is to generate networks that are too large networks which are sensitive to overfitting. An attempt to avoid this problem by using penalties in the fitness function for complexity, but this tends to generate ANNs that are too limited to be able to learn the functions required by the task (Liu&Yao, 1996). Therefore many researchers propose evolutionary methods that evolve both ANN architectures and weights.

2.4.1 Why evolution of ANN robot controllers?

According to Harvey *et. al.* (1996a) the designing of as complex system as cognitive control system for robot using the traditional method of divide and conquer (that is the most common approach to the design of complex systems) has at least the following problems:

> "- It is not clear *how* robot control system should be decomposed.
>
> - Interactions between separate sub-systems are not limited to directly visible connecting links between them, but also include interactions mediated *via the environment*.
>
> - As system complexity grows, the number of potential interactions between sub-parts of the system grows *exponentially*." (Harvey *et al.* 1996b, original emphasis)

Most robot controller problems can only use reinforcement learning as it is not possible to know the correctness of each decision.

> "The complications of the above problems has turned researchers to an alternative approach of using evolutionary techniques to incrementally evolve increasingly complex robot control systems" (Harvey *et al*., 1996a)

2.5 Related work

2.5.1 Evolution of ANNs

One of the most commonly used method for evolving ANNs is "Symbiotic, Adaptive Neuro-Evolution" or SANE (Moriarty, 1997). This method is unlike other currently used methods in that it evolves population of neurons instead of whole networks. One restriction on this system in its original form described in Moriarty (1997) is that it can only handle feed-forward ANN with one hidden layer. This has shown good performance on tasks that require no short-term memory (Moriarty, 1997).

Braun&Weisbrod (1993) describe another more conventional method called ENZO (Evolutiver Netzwerk-Optimierer). It evolves feed-forward network architectures but requires user specified maximal network architecture (number of layers and nodes in each layer). To speed up learning children inherit knowledge from parents by receiving a fraction of parents weights instead of a random weight. Braun&Weisbrod (1993) have shown some good results when applying ENZO to benchmark tests such as digit recognition.

Liu&Yao (1996) introduce a method that evolves both the architecture and the weights of ANNs. This method shows good results on simple benchmarks such as the 7 and 8 bit parity problem. Yao&Liu (1997) they describe a more complex version of the previously mentioned method that is based on evolutionary programming (EP). This method performs well on more complex benchmarks such as the medical diagnosis problem.

2.5.2 Evolution of robot controllers

Recently there has been much work done on the evolution of robot controllers. Some possible reasons for the increase of interest in this area can be found in section 2.4.1.

At the University of Sussex much research has been done on the evolution of robot controllers. Harvey *et al.* (1996a) gives an overview of this work, much of which has been on visually guided robots evolved using SAGA (Species Adaptation Genetic Algorithms). In Cliff&Miller (1995, 1996) co-evolution of pursuit and evasion is the issue, but the focus of the research is on the evolution process or as they describe it:

> "... does sustained competition really lead to smooth directional evolutionary progress, or to noisy, unreliable, fits and starts, or to endless cycling through different evolutionary unstable strategies? " (Cliff&Miller, 1995, p 1)

The result of this co-evolution are robot controllers that are specialized in competing the current best opponent but seem to lack generalisation when competing with older generations of opponents.

The method described in Gomez&Miikkulainen (1997) is an extension of SANE described in section 2.5.1. It is extended with enforced sub-population and delta-coding. The most important benefit of enforced sub-population is that it allows evolution of recurrent networks. A disadvantage is that it sets a limit on how many hidden units are used. Delta-coding is used to search for optimal modifications on current best solution (Gomez&Miikkulainen, 1997), which helps dealing with incremental evolution that Gomez&Miikkulainen (1997) argue is needed in the predator prey problem.

Another method is described in Fullmer&Mikkulainen (1991) and Moriarty&Miikkulaien (1995). Specific for this method is the marker-based encoding of the ANNs, which is inspired by the marker structure of biological DNA. This encoding removes most limitations on the architecture of the ANNs and gives as much control to the evolution as possible.

2.5.3 Comparison of methods for evolution of fully recurrent ANNs

Probably the best way to show how well a certain method performs is to compare it to other known methods, however the methods compared can have different advantages. This could imply that the test problem chosen can be better for one method than the other. The test problems also tend to be simple (e.g. pole-balancing or N-bit parity problem). Therefore a method that is good on simple things tasks but is unable to evolve ANN that can handle complicated task can seem to be better than network that is not as good at evolving simple ANNs but may evolve better ANNs for more complicated problems. The general factors compared are typically complexity of the resulting ANNs, time complexity and error rates.

Moriarty (1997) tests the SANE on the pole balancing benchmark and on simple robot controllers for the Khepera simulator. In pole balancing CPU time and pole balance attempts are compared to results of some other methods run on the same task. The mean CPU time used is 7.7 and mean pole balancing attempts are 900 over 50 runs, compared to CPU time of 9.8 and 2578 balancing attempts of GENITOR. Recurrent ANNs were not evolved as SANE in its original form does not handle recurrent ANNs. Gomez&Miikku-

lainen (1997) extend the SANE to handle fully recurrent ANNs and test the method on the prey capture task that requires the method to evolve short term memory. Their test shows very promising results as complex behaviour evolved using incremental evolution, however no comparison is made with other methods.

Braun&Weisbrod (1993) test ENZO on digit recognition problem and kinematics of a backing up truck within a delimited range of situations. Finally they tested on learning a scoring function for the endgame of Nine Men's Morris. The last problem the computation time was about one week on SUN 4-SLC workstation which gives some sense for the time complexity.

Liu&Yao (1996) test their method on the N-bit parity problem. An interesting result is that they were able to evolve ANN that solved the 8-bit parity problem with only 3 hidden nodes. In Yao&Liu (1997) they test the EPNet on the N-bit parity problem, medical diagnostic problems, the Australian credit card problem, and the MacKey-Class chaotic Time Series prediction problem. They also compare the result obtained from EPNet to the best hand designed ANNs (trial and error) and a ANN constructive algorithm called FNNCA. They base their comparison on number of connections, number of hidden nodes, number of generations, and error-rates (testing, mean, minimum and maximum).

3 Problem Definition

The design of complex robot controllers has proved to be a difficult task using conventional methods, and there is therefore an increasing interest in the evolution of robot controllers (see chapter 1).

This dissertation concerns comparison of two methods for evolving a robot controller that is able to perform tasks in a dynamic world. This task selected requires the agent evolved to have some form of short-term memory as things can disappear out of the range of the robot sensors. The robot controller must therefore be able to take advantage of sensor data from previous time step.

3.1 The task

The task selected to test these methods is the prey capture task described in Gomez&Miikkulainen (1997). This is a special case of the pursuit and evasion problem class. The predator moves through environment attempting to catch the prey while the prey attempts to avoid to be captured by fleeing. Gomez&Miikkulainen (1997) argue that scenarios of this kind are an interesting arena for the study of adaptive behaviour as they are ubiquitous in natural ecosystems and provide a simple objective that requires complex sensory-motor coordination with respect to the environment and another moving entity. The agent is successful when it is able to catch the prey within a predetermined number of time steps. The agent's sensors have only limited range and if the prey moves out of the sensor range the agent must remember where it was last detected. Gomez&Miikkulainen (1997) argue that directly evolving robot controllers does not give good results as the randomly generated controllers in the first generation have no possibility of getting near the prey therefore the selection is not based on the fitness as all initial agents get zero fitness. This results is an agent that moves around the environment bumping into the prey by chance, instead of employing a complex prey capture strategy. They therefore suggest incremental evolution, which they achieved by slowly increasing the speed of the prey and the number of initial movements given to the prey, i.e. gradually increase the difficulty of the task.

This problem is simple to describe and implement but the resulting robot controllers can be hard to analyze. The evolved ANNs will contain some mechanism allowing them to remember older states, i.e. short-term memory, resulting in a complex topology.

Another possible problem is to co-evolve the hunter and the prey. The difficulty in this task is that the fitness landscape is changing the whole time, a change often referred to as the Red Queen problem[1] (Cliff&Miller, 1995). Although this problem is interesting from an evolutionary point of view it results in agents that are specialized in competing with the other but lack generalization, i.e. they are not efficient when tested on advisories from few generations before.

---

1. The name Red Queen originates from Lewis Carroll's "Through the Looking Glass". The Red Queen was a living chess piece that ran but never got far as the landscape kept up with her.

3.2 The methods to compare

Here follows a brief description of the two methods chosen for comparison in this dissertation. The first of the methods has been tested on the predator-prey problem in an earlier work (Gomez&Miikkulainen, 1997). A detailed description can be found in chapter 4.

The first method which is SANE with enforced sub-populations plus delta-coding, this method is described in Gomez&Miikkulainen (1997) and is based on SANE (Moriarty, 1997) with two extensions: enforced sub-population and delta-coding. In Gomez&Miikkulainen (1997) this method is shown to be efficient in evolving robot controllers that can solve the task described in chapter 3.1. The enforced sub-population extension allows the evolution of recurrent networks and speeds up the evolution. On the other hand the decision of the number of hidden units must be predetermined and is therefore out of the control of evolution. The advantage of delta-coding is that it can search for optimal modifications of the current best solution Gomez&Miikkulainen (1997).

> "Delta-Coding explores the hyper-space in a "neighborhood" around the previous best solution." (Gomez&Miikkulainen, 1997, p 7)

Therefore Gomez&Miikkulainen (1997) argues that this is helpful in incremental evolution.

The second method is marker-based encoding, and is described in detail in Fullmer&Mikkulainen (1991) and Moriarty&Miikkulainen (1995). This encoding is loosely based on the marker structure of natural DNA. Unlike the first method, marker-based encoding allows all aspects of the ANN, including the number of nodes and connectivity, to be under evolutionary control. When the restrictions on the network's design are minimal, the genetic algorithm has maximal search space which increases the chances of evolving the optimal network for the problem. There are currently no extensions to improve this method, but when combining this method with incremental evolution it might be a good idea to extend it with delta-coding because of Gomez&Miikkulainen's (1997) arguments.

In Fullmer&Mikkulainen (1991) this method was successfully applied on a robot controller problem that requires some memory in a grid world simulation. This method is therefore a good candidate for solving the task at hand with much simpler implementation than SANE with enforced sub-population and delta-coding.

3.3 Comparison

Yao&Liu (1997) base their comparison on the number of connections, number of hidden nodes, number of generations, and error-rates (see section 2.5.3).

In this dissertation the number of input-nodes and output-nodes is static and predetermined. The number of hidden nodes are predetermined in enforced sub-population plus delta-coding and the evolved ANNs are also fully connected. however the number of connections and nodes can be compared as marker-based encoding can evolve any topology so that gives comparison of the complexity of the evolved ANNs. If one method uses much fewer generations to evolve an agent that can handle the hardest difficulty of the task, it requires less computational power than the other and therefore gets more runs. It is very likely that enforced sub-population plus delta-coding needs fewer generations to evolve an ANN but it could be better with different number of neurons than used in the run. Marker-based encoding on the other hand gives the evolution control of the number of nodes in the network, and therefore probably requires more generations to arrive to a comparable solution. Therefore the enforced sub-population plus delta-coding could be run more times with different number of neurons to make the comparison fair with respect to the computational power. This can be done as both methods are considered to require similar computational power for each generation (see chapter 4.4), that also gives the possibility of using number of generations used to reach a solution to compare computational power used.

As reinforcement learning is used there are no error-rates to compare but as both methods use the same fitness function the resulting ANNs can be put to a number of extra runs and the average fitness can be compared.

Based on the results Gomez&Miikkulainen (1997) where individuals capable of capturing prey of difficulty 7 were evolved in less than 200 generations and that marker-based encoding is expected to need more generations than that, the maximum number of generations is chosen to be 10000 generations. If individual capable of handling prey of difficulty 7 is not reached with in these limits then the method fails the task.

The methods will be compared on the basis of the number of nodes and connections in a representative evolved ANN for each method. Also the average fitness of the same ANN when confronted with a number of extra test runs on the simulator, even adding noise to the input values of the input nodes can be compared to investigate the robustness of the evolved ANNs.

Marker-based encoding should be capable to perform equally well or even better on this problem as there are no limitations on the topology, and with delta-coding the computational power needed is similar to SANE with enforced sub-populations and delta-coding.

Based on the arguments above, the following hypotheses are made:

- Marker-based encoding is as capable to perform equally well or even better on this problem
- With delta-coding the computational power needed for marker-based encoding is similar to SANE with enforced sub-populations and delta-coding

4 Method

Here the two methods chosen in this dissertation are described in detail.

4.1 Enforced sub-populations plus delta-coding

This method is described in Gomez&Miikkulainen (1997) and is based on SANE (Moriarty, 1997) with two extensions: enforced sub-populations and delta-coding.

SANE evolves populations of neurons rather than complete networks. The neurons are combined to form a hidden layer in a feed-forward network which is then evaluated by its performance on the problem at hand (see section 4.3.3 for the fitness function). The process can be described as follows:

Initially the number of hidden units is specified and the first population of neuron chromosomes is randomly created. The hidden units are fully connected and each weight is in the range from -6.0 to 6.0 (as in Gomez&Miikkulainen (1997)).

Next, a set of randomly selected neurons form a hidden layer of a network and the network is evaluated on the task at hand resulting in a fitness value. The fitness is added to the cumulative fitness of each individual neuron of the network. This is repeated until each neuron has gone through some predetermined number of evaluations on average. Here three evaluations on average were used to counter evolutionary noise, as in Gomez&Miikkulainen (1997).

The average fitness of each neuron is then calculated and neurons are ranked by fitness. Randomly selected neurons in the top quarter of the rank list are recombined with a higher ranking neuron using 1-point crossover and mutation with 10% probability. The resulting offspring then replace the low ranking half of the rank list.

This process is iterated, calculating the fitness of the new generation and creating a new one. This is continued until the average fitness of the top nodes in the population reaches a selected value, which can be set for each fitness step. In this dissertation the values were 40 for level 0, 50 for level 1, 60 for level 2 and 80 for levels 3 through 7.

Enforced sub-population (ESP) is an extended version of the SANE method but instead of using one population containing all the neurons, the ESP allocates a separate population for each of the neurons in the hidden layer. This speeds up SANE as sub-populations that gradually form in SANE are already divided up, and no recombination can occur between two different sub-populations. Another reason for the increase in speed is that the networks used for evaluation always contain one neuron from each sub-population evaluating how well each neuron performs in cooperation with all the other sub-populations.

The most important advantage of ESP over SANE is that it gives the possibility of evolving fully recurrent networks. The sub-population architecture of ESP causes the neuron's recurrent weight r_i to always be associated with neurons from sub-population S_i. As the sub-populations specialize, neurons evolve to expect the kind of neurons that they are connected to (Gomez&Miikkulainen, 1997). As the prey gets a number of initial moves the agent must remember where the prey was sensed last to be able to take of in the right direction after the prey when it is allowed to move. This requires a recurrent ANN as mentioned in chapter 2.1.4.

The idea of delta-coding is to search for optimal modifications of the current best solution (Gomez&Miikkulainen, 1997). It is starts by saving the current best solution and then initializing a population of new individuals called delta-chromosomes. The new delta-chromosomes have the same number of genes as the current best solution and consist of delta-values that represent differences from the current best solution. The delta-values are chosen randomly with highest probability of a small change using Gaussian (normal) distribution[2] with 0 mean and standard deviation 1. Then the delta-chromosomes are evolved using the same mechanisms as in SANE with delta-coding but using mathematical crossover[3] instead of one point crossover.

The purpose of this implementation is to have a method that has previously been applied on this problem with claimed good results. Then comparing to the results of this method is important to get some insight into the performance in a larger perspective.

The SANE implementation is based on descriptions in Gomez&Miikkulainen (1997) and Moriarty (1997). Enforced sub-populations as well as delta-coding are implemented according to the descriptions found in Gomez&Mikkulainen (1997).

4.2 Marker-based encoding

Marker-based encoding is described in Fullmer&Miikkulainen (1991) and Moriarity&Miikulainen (1995). This encoding is loosely based on the marker structure of natural DNA. This approach allows all aspects of the network, including the number of nodes and its connectivity, to be under the evolutionary control. By removing all restrictions on the network design the search space for the genetic algorithm is as large as possible, thus maximizing the chances of evolving an optimal network.

> "The key to feature of the marker-based encoding scheme is to use marker values to section off the working areas of the genetic material." (Fullmer&Miikkulainen, 1991, p 3)

Markers are used to separate node definitions. Each definition contains all the information needed to carry out the nodes computations. The structure is shown in Figure 5, the chro-

2. In Gomez&Miikkulainen (1997) Cauchy distribution with α-value 0.3 and mean 0 (The mean of a Cauchy distribution actually does not exist strictly, but as the distribution is symmetric then it is generally taken to be the value where the probability is highest). The main difference between normal and Cauchy distribution is that Cauchy has infinite variance while normal distribution has σ^2 as variance. Gaussian (normal) and Cauchy distribution look the same except Cauchy gives little more probability of values further from the mean.

3. Mathematical crossover is implemented so that child DNA gets the value of A*parent1DNA + B*parent2DNA, where A and B sum to unity. Here A was chosen 0.25 and B 0.75, these values were chosen as they were used by Gomez&Miikkulainen (1997).

mosome is a string of integers that are in the beginning randomly generated in the range -128 to 127.

<start><key><initial_value><k_1><l_1><w_1><k_2><l_2><w_2>.......<k_n><l_n><w_n><end>

Where:

<start>	is a node start marker.
<key>	is a node identification value used in the connections
<initial_value>	is the output value of the node prior to first activation
<k_i ><l_i ><w_i >	are connection specifications: k_i specifies type of connection, to input-,output- or hidden-unit. l_i specifies the label of the unit to which the connection is. w_i specifies the weight of the connection.
<end>	is a node end marker.

Figure 5: Structure of a node definition.

A node definition contains the identification of the node, initial value, and a list specifying its input sources, output nodes, and connection weights. The neuron may get its input from other neurons, sensors, or form its own output. Because of the latter this encoding scheme supports fully recurrent ANNs. Start and end markers are identified by their absolute values: if the value modulo a certain predetermined number[4] is 1 then it is a start marker, if the result of the modulus is 2 then it is a end marker. In Fullmer&Miikkulainen (1991) this value is 15 and in Moriarity&Miikulainen (1997) it is 25. In this dissertation the higher value is used (i.e. 25) to decrease the number of neurons in an attempt to make the resulting ANN comparable to SANE with 5 hidden nodes. Integers between the end marker and the start marker are ignored (sleeping DNA). The chromosome is implemented as a linear list but is treated as a continuous circular entity so that node definition may begin near the end of the list and continue at the beginning of the same list. A standard genetic algorithm is used including two-point crossover and mutation, the mutation is done by adding a random value to a each number in the integer vector with 0.4% probability. If the mutated value is out of range (-128, 127) then it is warped around (e.g. 130 becomes -3 and -140 becomes 12).

One property of marker-based encoding is that multiple synapses can form from one node to another. These can be summed to one synapse decreasing the number of synapses in the

4. The number controls the density of the node definitions, the lower value of the modulus number the higher probability of may nodes.

ANN. In Fullmer&Miikkulainen (1991) this method is used to evolve an agent capable of performing simple tasks requiring memory in a different grid world simulation.

Marker-based encoding is described in Fullmer&Miikkulainen (1991) and Moriarity&Miikulainen (1995). The implementation here are based on those papers. The purpose of this implementation is to compare the marker-based encoding to SANE with enforced sub-populations with delta-coding.

In this dissertation a variation of marker-based encoding with delta-coding is also tested to investigate some of the implications of what delta-coding has on marker-based encoding. It can be compared to the plain marker-based encoding and it can also be compared to enforced SANE with sub-populations with delta-coding to get an more even comparison of SANE with enforced sub-populations and marker-based encoding.

In this implementation delta-coding had to be changed to handle integer weights in the range -128 to 127, so Gaussian random values were used with 0 mean and standard deviation 25. To try to get only the good parts of the parent genes a version of arithmetic crossover that is here referred to as two point arithmetic crossover was used. This mechanism can best be described with a pseudo code shown in figure 6.

```
input: parent1, parent1
output: child1, child2

      for i:=0 to random_position1 do

          child1DNA[i] := 0.25*parent1DNA[i]+0.75*parent2DNA[i]
          child2DNA[i] := 0.25*parent2DNA[i]+0.75*parent1DNA[i]

      for i:= random_position1 to random_position2 do

          child1DNA[i] := 0.25*parent2DNA[i]+0.75*parent1DNA[i]
          child2DNA[i] := 0.25*parent1DNA[i]+0.75*parent2DNA[i]

      for i:= random_position2 to size_of_gene do

          child1DNA[i] := 0.25*parent1DNA[i]+0.75*parent2DNA[i]
          child2DNA[i] := 0.25*parent2DNA[i]+0.75*parent1DNA[i]


return child1 and child2
```

Figure 6: Pseudo code for two point arithmetic crossover

4.3 Simulator

An overview of the simulator can be seen in figure 8. The simulation starts with the prey getting a number of initial moves and the agent receiving its perceptions but not allowed to make any moves. When the number of initial moves (defined by the difficulty level) is reached then moves are in turns, i.e. first the prey moves and then the agent makes its move. If the agent or the prey tries to move into a wall then they lose the move, i.e. remain at the same position. The size of the simulated environment is 24x24.

The interactions between the environment and the other modules are shown in figure 7.

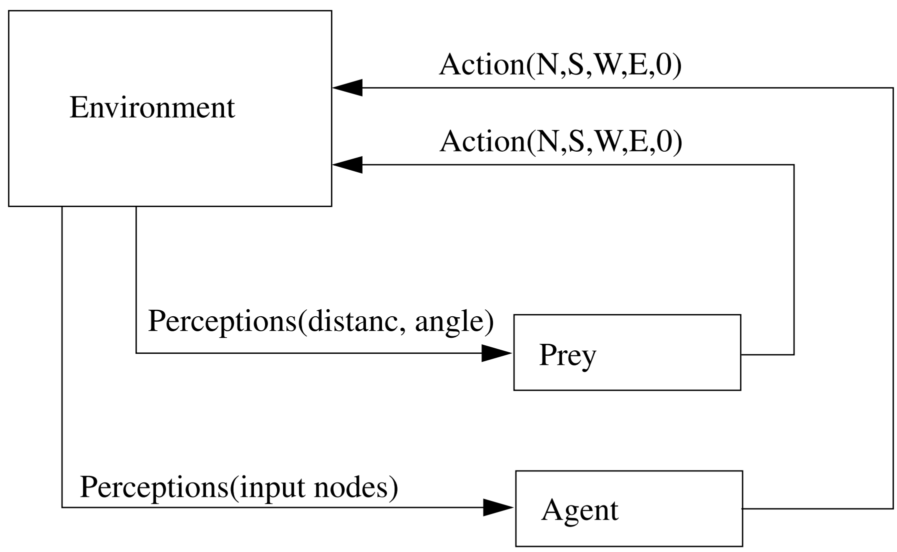


Figure 7: Interaction between different modules

4.3.1 The agent

The agent is controlled by ANN consisting of neurons with sigmoid activation functions. At each time step every node receives input from both the input layer and from the other units that are connected to it.

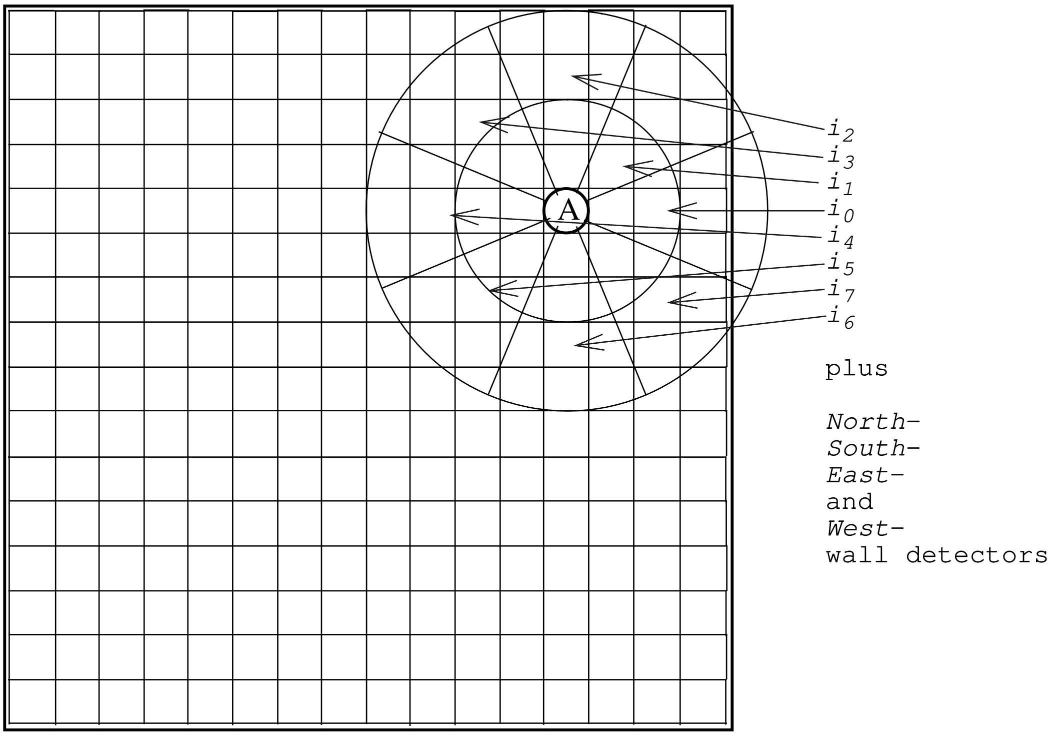


Figure 8: The input nodes of an agent

The input nodes can be seen in figure 8. Where i_i are sensors for detecting the prey, *C* is set when the prey is close (within the inner circle) and the wall detectors are activated in accordance with the distance from the walls that are in sensory range.

Gomez&Miikkulainen (1997) set the prey sensor to 1 if the prey is within sensory range. The *C* node is used to give some estimation of distance, i.e. how close the prey is. In this dissertation the activation of the prey sensors depend solely on the distance to the prey. This simulates better a real world scenario when the sensor inputs are mapped directly to input nodes, and makes the C input node superfluous. The sensory range is chosen to be 5 as in Gomez&Miikkulainen (1997). The agent can move in one of four directions or stand still. Standing still allows the agent to change internal state, as recurrent ANNs can have internal states. Noise is introduced by adding a random value to the inputs. The random values range is within (-noise_level, noise_level), and as the sensory inputs give values from 0 to 1 then the noise value is in percents.

4.3.2 The prey

The prey uses probabilistic functions to decide on which direction to move to. The probability of a move in each direction is dependent on the relative position of the agent so moving from the agent is most probable, but an irrational move is also possible. The algorithm is described in Gomez&Miikkulainen (1997) and is shown in EQ1 - EQ4 below, where *angle* is the angle between the direction of the action and the direction from the prey to the agent, and *dist* is the distance between the prey and the agent (figure 9).

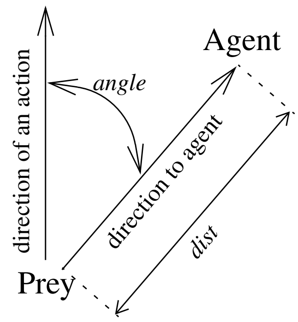


Figure 9: Prey perceptions

$$prob(A_X) = \frac{P_X}{P_N + P_S + P_E + P_W} \quad \text{(EQ 1)}$$

$$P_X = e^{(0.33 \cdot W(angle) \cdot T(dist))} \quad \text{(EQ 2)}$$

$$W(angle) = \frac{180 - |angle|}{180} \quad \text{(EQ 3)}$$

$$T(dist) = \begin{cases} 15 - dist & if \quad dist \le 4 \\ 9 - \frac{dist}{2} & if \quad 4 < dist \le 15 \\ 1 & if \quad 15 < dist \end{cases} \quad \text{(EQ 4)}$$

As described in section 3.1 the prey must be fast to force the robot controller to use the knowledge of previous state in deciding on an action. In the beginning of the evolution the randomly generated agents have no idea of how to catch a fast prey. Therefore the fitness will not contribute enough information on which to base the selection for reproduction. This is why the prey must be made to move slowly in the beginning and increases its speed as the evolved agents get better. The method chosen to accomplish this is to include the possibility of standing still and have the probability of this high in the beginning and then decrease in steps as the agents get better in accomplishing the task.

The decision to increase the prey difficulty can be based on two facts: when the fitness values from generation to generation start to level off or when the fitness has reached a certain value. Here the latter is chosen as in the first method: the population is converging to an agent that is good on current prey but maybe lacks some attributes that are necessary later. At convergence the population looses variation that can be valuable in future evolution. The fitness threshold is calculated from the mean fitness of four best individuals in a population, prey speed and preys initial moves[5] for each difficulty level is shown in table 1.

TABLE 1.

Difficulty	Prey Speed	Preys initial moves	Fitness threshold
0	0	0	40
1	0	2	50
2	0	3	60
3	0	4	80
4	0.3	4	80
5	0.6	4	80
6	0.8	4	80
7	1.0	4	80

4.3.3 Fitness function

The fitness function is the same as described in Gomez&Miikkulainen (1997). If the prey is captured within a predetermined *N* number of steps, the fitness is increased with 1 and another attempt is allowed. Here *N* is defined 60 as that was the value used by Gomez&Miikkulainen (1997) (Gomez, personal communication). If the prey is not captured within the predetermined maximum number of steps then the acquired fitness is the given fitness for the ANN being tested. If the fitness reaches 100 the test is aborted and the given fitness is 100. In other words, if the agent caches the prey 100 times, each time within the given maximum time then the method is considered a success. After 10000 generations without an adequate solution (finding an agent that has fitness over 80 when tested at difficulty level 7) the method is considered a failure.

4.4 Individuals in a population

The two approaches require different amount of individuals in their respective population. The reason for this is that in each sub-population of enforced sub-population with delta-coding needs adequate number of individuals whereas in marker-based encoding only one population is required. Therefore the marker-based encoding does only need as many individuals in its entire population as enforced sub-populations needs in one sub-population. On the other hand in enforced sub-populations more than one node is evaluated at a time so CPU usage depends mostly on the number of individuals in a sub-population although there are a lot more recombinations and mutations. Each sub-population is given the same number of individuals in enforced sub-populations, as the entire population in marker-based encoding. This ensures that each generation requires the same amount of evalua-

5. Preys initial moves are in speed 1.0

tions. Therefore the number of generations used gives information that can be used to compare CPU power used.

5 Results

Here the results of representative runs are presented. The difficulty steps are different between runs, so an average of many runs can not be used. The time for testing only allowed methods to be tested on of 3 runs if they were a failure.

5.1 SANE with enforced sub-populations

SANE with enforced sub-populations failed to produce a satisfactory results as it only made it to difficulty 4 in 10000 generations in the representative run. Other runs ranged from not passing difficulty 0 to getting to difficulty 5. This deems the method as a failure. Figure 10 shows the fitness values of the evolution in a representative run. The vertical dotted lines show when difficulty level increases. The difficulty levels are marked with "d:*n*" in the figure, where *n* is the difficulty level.

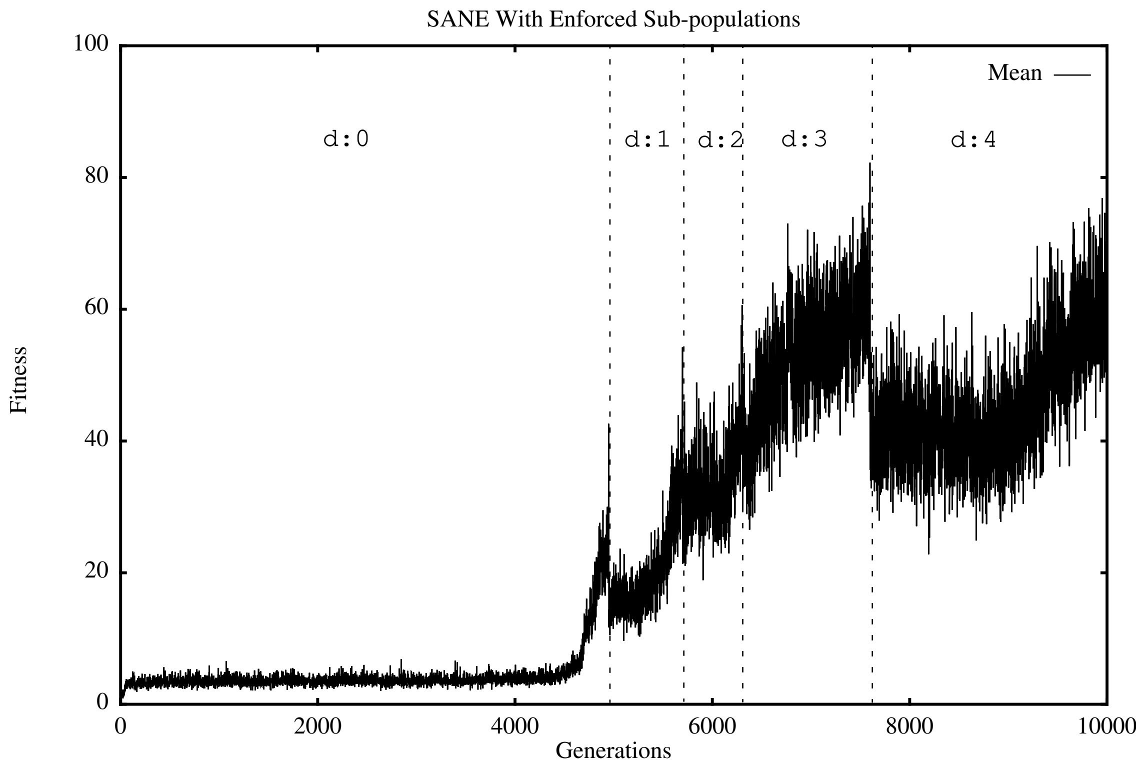


Figure 10: Fitness values of generations in SANE with enforced sub-populations

As this method failed to get to difficulty level 7 tests with noise are not run. The evolution is done with no noise and there is no reason to believe that performance on difficulty level 7 with noise would give any better result.

The generations for each difficulty level is shown in table 2 below.

TABLE 2.

Generations	Difficulty
0 - 4949	0
4949 - 5697	1
5697 - 6301	2
6301 - 7598	3
7598 - 10000	4

5.2 SANE with enforced sub-populations and delta-coding

Surprisingly this method failed to produce satisfactory results as it only made it to difficulty 4 in 10000 generations in the representative run, other runs ranged from getting to difficulty 1 to making it to difficulty 6. This deems the method as a failure. Figure 11 shows the fitness values of the evolution of a representative run, the vertical dotted lines show when difficulty level increases. The difficulty levels are marked with “d:*n*” in the figure, where *n* is the difficulty level.

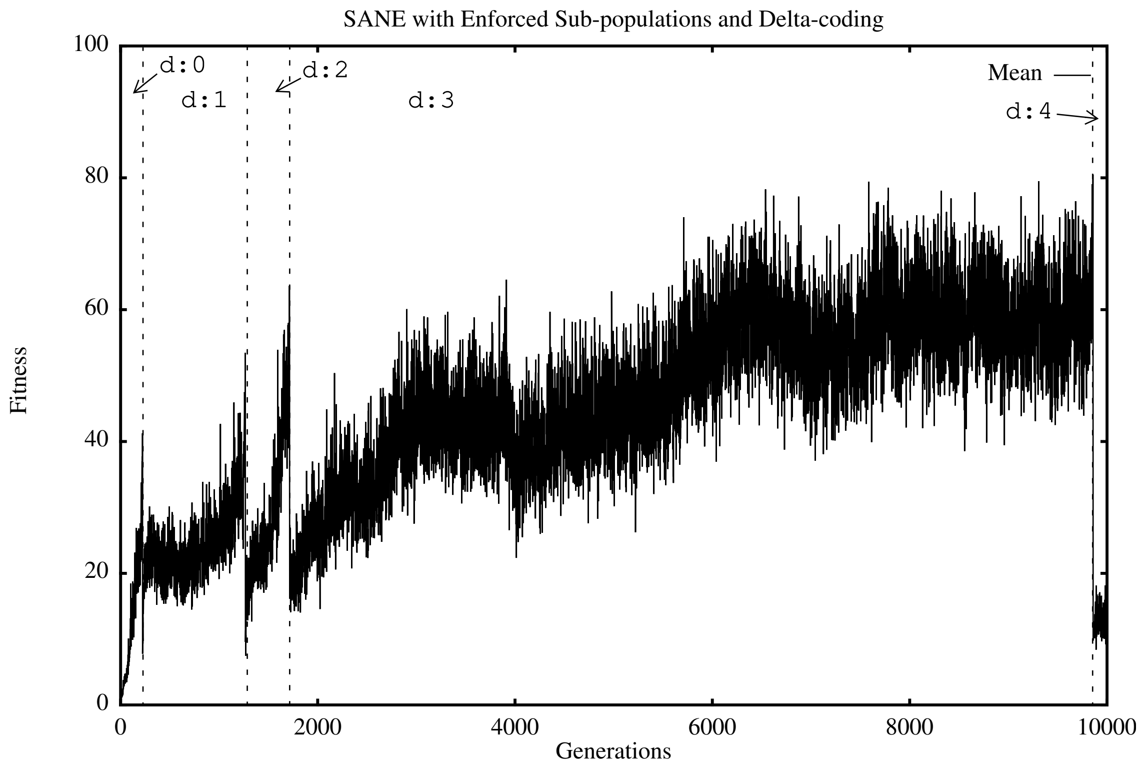


Figure 11: Fitness values of generations in SANE with enforced sub-populations and delta-coding

As this method also failed to get to difficulty level 7 tests with noise are not run for the same reasons as in previous section.

The generations for each difficulty level is shown in table 3 below.

TABLE 3.

Generation	Difficulty
0 - 223	0
223 - 1266	1
1266 - 1714	2
1714 - 9859	3
9859 - 10000	4

5.3 Marker-based encoding

Marker-based encoding produces satisfactory results in only 493 generations in the representative run which is much faster than expected. Other runs needed from 427 to 579 generations to produce satisfactory results. Figure 12 shows the fitness values of the evolution of a representative run, the vertical dotted lines show when difficulty level increases. The difficulty levels are marked with "d:*n*" in the figure, where *n* is the difficulty level.

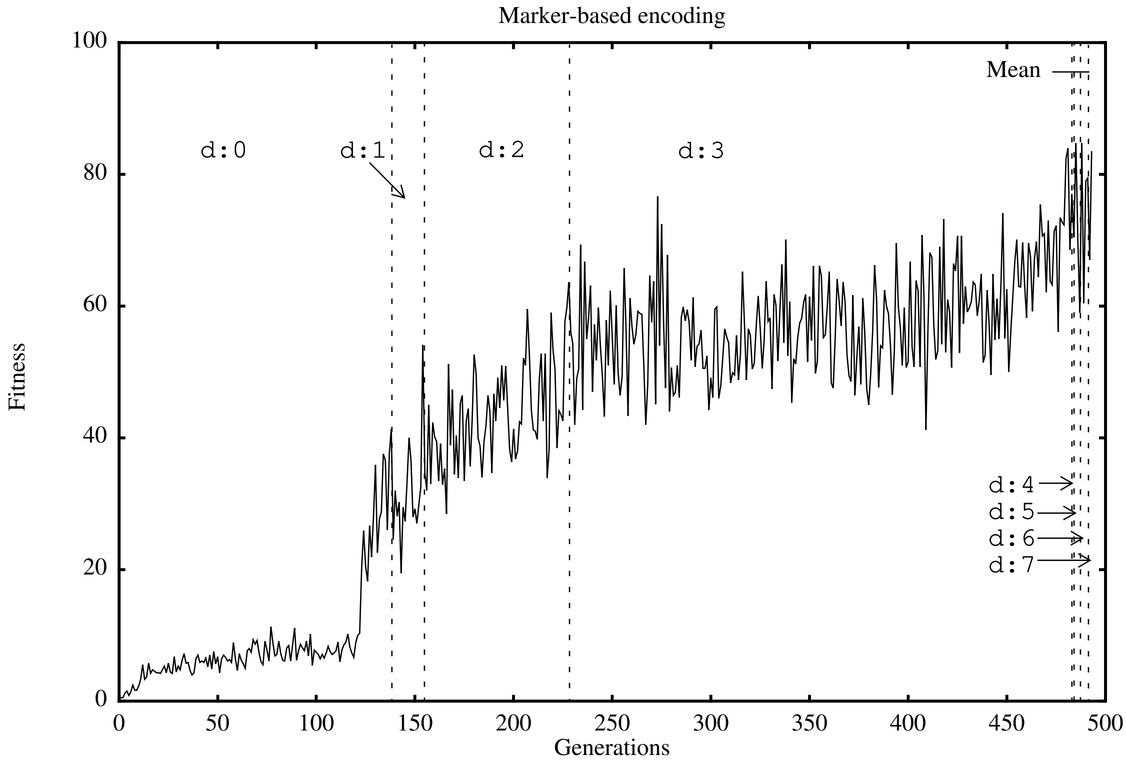


Figure 12: Fitness values of generations in marker-based encoding

The generations for each difficulty level is shown in table 4. below.

TABLE 4.

Generation	Difficulty
0 - 137	0
137 - 154	1
154 - 228	2
228 - 480	3
480 - 481	4
481 - 485	5
485 - 488	6
488 - 493	7

To analyze the evolved ANNs all superfluous synapses[6] were removed. This procedure was done manually as the time did not allow for more coding to be done (it can however be automated). The ANN, which evolution is shown in figure 11, has 16 hidden nodes and 81 synapses after eliminating duplicate synapses at every node. Table 5 shows the results from running additional tests on the ANN with additional noise the fitness is a mean fitness from 100 tests. The fitness starts to decrease between 20% and 30% noise.

TABLE 5.

Fitness	Noise (0.00 to 1.00)
42.42	0.01
44.99	0.02
43.88	0.03
45.75	0.04
43.90	0.05
59.68	0.10
39.08	0.20
18.89	0.30
11.51	0.5
5.57	1.0

5.4 Marker-based encoding with delta-coding

Surprisingly this method used much more generations to produce a satisfactory results than just marker-based encoding as it used 8917 generations to produce a satisfactory result in the representative run. Other runs ranged from needing 6874 generations to only making it to difficulty 6. Figure 13 shows the fitness values of the evolution of a represent-

6. In marker-based encoding many synapses can form from one node to another. This makes a model that simulates the natural connections between individual natural neurons while artificial neural network's neurons are more of a simulation of clusters of natural neurons. Therefore the weights of the multiple synapses can be added and one synapse used instead of many, diminishing computation need to calculate an output from the ANN.

ative run for this method. The vertical dotted lines show when difficulty level increases. The difficulty levels are marked with “d:*n*” in the figure, where *n* is the difficulty level.

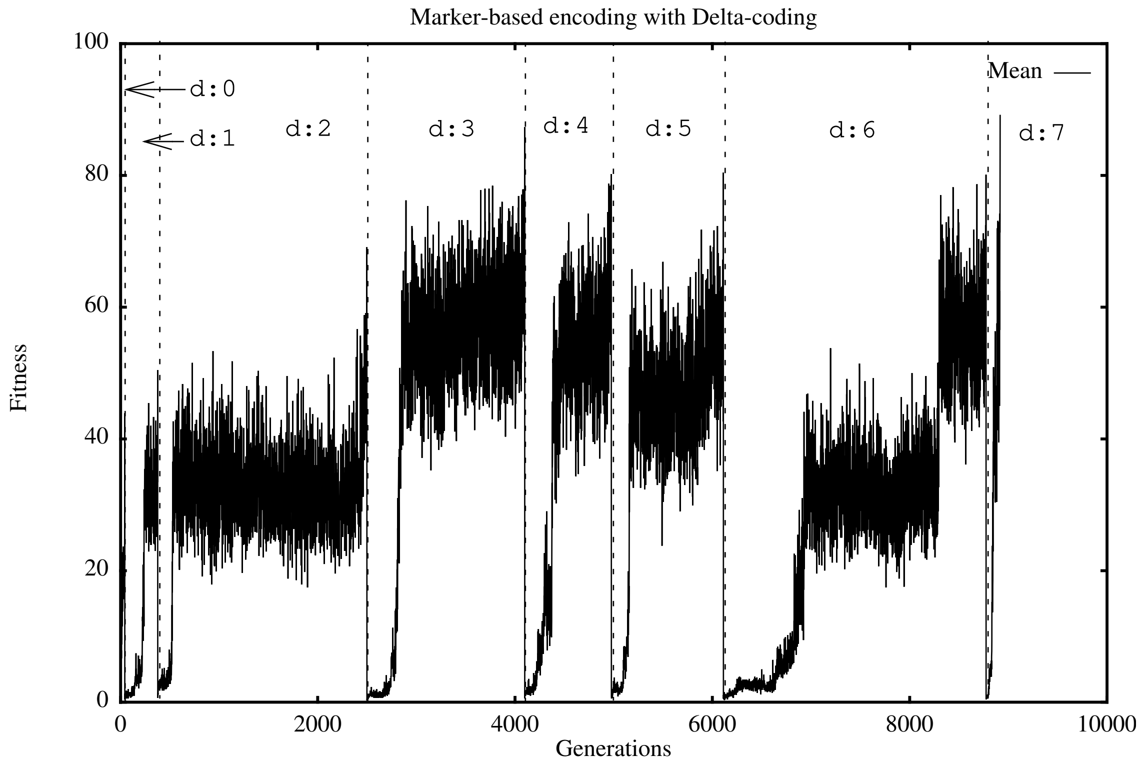


Figure 13: Fitness values of generations in marker-based encoding with delta-coding

The generations for each difficulty level is shown in table 6 below.

TABLE 6.

Generation	Difficulty
0 - 43	0
43 - 378	1
378- 2489	2
2489 - 4098	3
4098 - 4978	4
4978 - 6112	5
6112 - 8775	6
8775 - 8917	7

The resulting ANN contains 24 nodes and 92 synapses after removing duplicate synapses the same way as mentioned in previous section. Table 7 shows the results from running additional tests on the ANN with additional noise the fitness is a mean fitness from 100 tests. The fitness here starts to decrease after 30% noise.

TABLE 7.

Fitness	Noise (0.00 to 1.00)
42.42	0.01
44.99	0.02
43.88	0.03
42.97	0.04
41.51	0.05
40.65	0.10
38.82	0.20
40.36	0.30
38.93	0.5
8.63	1.0

6 Discussion and conclusion

6.1 SANE with enforced sub-populations

This test is mainly done to compare the difference delta-coding makes on SANE with enforced sub-populations. Before this test the results were expected to be worse than SANE with enforced sub-populations and delta-coding. A surprising result is however the long time (4949 generations) it takes to find an agent that can handle the simple problem of difficulty 0. One possibility is that there has been bad random values when initiating first generation with random values so the evolution becomes a random search using the mutation operator. The other runs done using this method did however show that eventually this method comes to a grinding halt on some local maximum. Another possibility is that the reproduction and selection is to aggressive, the parents are chosen from top 25% of the population and the children replace the bottom half of the population. This causes the evolution to throw away half of the population as the first generations get very low fitness values for all individuals, so the ranking of early populations is mostly random. This could possibly be solved by creating a much larger first generation and then decrease the number of individuals as the number of generations increases. The larger number of individuals in first generation would ensure larger variation in the population and therefore increase the possibility of getting individuals with good potential early in the evolution. Another possibility of tackling this problem is to allow delta-coding to set in when fitness values start to level off, indicating that the population is converging to an unwanted local maximum.

6.2 SANE with enforced sub-populations and delta-coding

In this implementation delta-coding starts when SANE with enforced sub-populations has found a solution to difficulty 0. The first surprise is that here fitness of 40 on difficulty 0 is reached in only 223 generations (usually within 2000 generations), compared to 4949 generations in SANE without delta-coding. As delta-coding does not start until at difficulty 1, this adds to our suspicion that the random values in the beginning of evolution are very important to the speed of evolution. When difficulty of 1 is reached delta-coding starts looking for optimal chromosome modifications of the currently best agent, then it takes about 1000 generations to evolve an agent that can handle difficulty 1. Then delta-coding restarts with new populations and uses only about 500 generations to evolve an agent capable of handling difficulty 2. Then the problem begins, it takes about 7000 generations to evolve agent capable of handling difficulty 3. When the difficulty is increased to 4 the fitness values take a dive but there is to little time left to evolve further. 10000 generations are reached with fitness 10-18 on difficulty 4.

6.3 Why so unexpected results?

This result is quite different from the ones presented in Gomez&Miikkulainen (1997), where good individuals handling difficulty 7 are found within 200 generations. Possible reasons for this are:

6.3.1 Changes in the simulator

The difference between the task used in this dissertation and the task used in Gomez&Miikkulainen (1997) is the values given to the input nodes and the absence of the C node. In Gomez&Miikkulainen (1997) the input nodes get the value 1 if the prey is within sensory range and else it has the value 0. To give the agent some estimation of the distance to the prey a special input node called C-node is used, it is set to 1 if the prey is within the inner half of the sensory range. In this dissertation, to make the task a little more like actual robot controller task then the sensory inputs were given values depending on the distance to the prey which simulates the activation of actual distance sensors that give higher values when objects are close. This change makes the C input node redundant, so it is not used in this dissertation. It is possible that this causes the task to be harder for the agent to handle. For detailed description of the task refer to sections 3.1 and 4.3.

6.3.2 Predetermined number of nodes

As can be seen on the results, the number of hidden nodes used in the marker-based encoding solutions is much higher than the number of nodes chosen in SANE with enforced sub-populations and delta-coding. The latter is based on the results of Gomez&Miikkulainen (1997) where 5 hidden nodes were used. This implies that the changes in the task require a much more complex ANN. This shows one disadvantage of the SANE with enforced sub-populations and delta-coding, namely that the number of nodes must be chosen manually and a small change in the task can require very different ANN.

6.3.3 Implementation differences

Another reason could be implementation differences, even though the implementation was done according to description in Gomez&Miikkulainen (1997) and with help from Faustino J. Gomez who kindly answered questions about the implementing details that were unclear in the paper.

The coding is described in section 4.1, the difference from the coding by Gomez&Miikkulainen (1997) is the calculation of output values and the initiation of delta values for delta coding.

In Gomez&Miikkulainen (1997) the activations of the hidden nodes is used to decide output values, i.e. one hidden node for each direction and one for not moving then the node with the highest activation deceits if and where to move. In this dissertation the hidden nodes are not previously assigned one direction. Instead there are 5 output nodes that are assigned the actions, which can make the evolution harder and therefore more generations can be expected than in Gomez&Miikkulainen (1997).

6.3.4 Delta-coding

In Gomez&Miikkulainen (1997) the delta values are generated using Cauchy distribution with mean 0 and α-value 0.3. In this dissertation the delta values are generated using Gaussian distribution with mean 0 and standard deviation 1. When using Gaussian distribution there is less possibility of getting large values which could contribute to the bad results when using delta-coding.

6.3.5 Improved delta-coding

It is clear that the time wasted on being stuck at difficulty 3 is a big contribution to the failure of this method. Could it be the case that better results can be achieved if delta-coding is allowed to create a new generation when the fitness values start to level out instead of when the difficulty is increased? To test this a simple change was made to the implementation that uses very simple mechanism to test if the fitness is levelling out and when that happens, new delta generation is created. The test function takes the mean value of the previous 10 fitness values and compares to the 10 before that. If the difference is below 2 and there are at least 20 generations since last delta population was created, then new delta population is created. The results, which can be found in Appendix A.1, show that the modifications above do not improve the method and produce agent that can handle difficulty level 7. The modifications were tested with few different parameter settings but that did not seem to improve the results. The function used to determine the decision to start a new delta population might need to be more complex, taking into account more information about the evolution and the fitness values over a longer period based on integration.

These results show that the SANE with enforced sub-populations with or without delta-coding as implemented in this dissertation using 5 hidden nodes does not handle the test problem chosen for comparison, however each coding is about 1400 lines of code so the absence of errors can not be guaranteed.

6.3.6 Number of nodes and initiation of delta values

The differences discussed in chapter 6.2 give some indication of how SANE with enforced sub-populations and delta-coding can be changed to perform better on this task.

To investigate these, additional experiments were carried out using 16 and 18 hidden nodes instead of the 5 in the first runs and using Gaussian distribution with mean 0 and standard deviation 6 (instead of 1) to give more variation to the delta values.

Just increasing the deviation to 6 did not improve the results. When then number of hidden nodes was increased to 18 then a solution was found in 4662 generations. But when using both more hidden nodes and deviation 6 the results were much better, evolving individuals capable of handling difficulty 7 in just 2768 generations with 16 hidden nodes and 1787 generations with 18 hidden nodes. The results can be found in Appendix B. With 16 hidden nodes the number of synapses is 528 and 630 with 18 hidden nodes. This shows that the complexity of these solutions is considerably higher than complexity of marker-based encoding solutions.

When using standard deviation 6 instead of 1 the evolution is faster. This result suggests that the use of Gaussian distribution instead of Cauchy distribution may contribute to slowing down the delta-coding.

These results show that SANE with enforced sub-populations and delta-coding is still slower than marker-based encoding. The reason for the difference between these results and those of Gomez&Miikkulainen (1997) is therefore presumably the increased complexity of the task and ANN.

This also shows that a big difficulty in using SANE with enforced sub-populations and delta-coding, namely that the number of hidden nodes is predetermined. As shown here a small increase in the difficulty of a task can require large change in complexity of the ANN.

6.4 Marker-based encoding

This method as opposed to others shows surprisingly better result than expected. In chapter 3.3 the hypothesis was that this method would produce equally good agents as SANE with enforced sub-populations and delta-coding, but would take much longer time. The actual results are that the evolution time is much shorter than the other methods and a agent capable of handling difficulty 7 is found in only 493 generations. As table 5 shows the representative agent endures noise levels up to 20% and has comparable complexity to SANE (81 synapses and 16 nodes compared to 110 synapses for 5 hidden nodes in fully connected SANE). It is also noteworthy that there are minor decrease in fitness when difficulty is increased. The final difficulty steps take only a few generations each, which suggests that a general solution that has short term memory is found early in the evolution. These results imply that delta-coding is not necessary for marker-based encoding to perform well, however the hypothesis was not only that marker-based encoding could produce equally good agents as SANE with enforced sub-populations and delta-coding, but also that delta-coding would increase the speed of the marker-based encoding so that the evolution time would be comparable also.

6.5 Marker-based encoding with delta-coding

This run is done to try to speed up marker-based encoding and should be very efficient on incremental evolution. This however is not the case as it takes 8917 generations to find a adequate agent handling difficulty 7. The noise tolerance is however a little better than of the ANN from marker-based encoding with delta-coding. In figure 11 it can be seen that fitness drops for every difficulty step and delta-coding soon helps the agent to regain the same fitness as before the difficulty step. At this point it seems to carry out a random search with the mutation operator. This could be countered with same modifications as shown in section 6.2. There new delta generation is not started when difficulty is increased but when fitness levels are starting to level out. The results from these runs can be found in Appendix A2. These results show that this method does increase the speed of the evolution of marker-based-encoding with delta-coding, it is very likely that a more complex function is needed for checking if the fitness is levelling off, and ensure that new delta population is not started too soon.

6.6 Conclusion

In this dissertation a comparison of two evolutionary methods evolving ANNs for robot control is made. The robot control task selected is the hunter and prey task, where the agent is to act as a predator trying to capture a prey. This task requires an agent that has some kind of short term memory, thus requiring a recurrent ANN. The start state of the simulations of the task is random, this requires that the agent evolved must be generalized. Robustness is investigated by testing the resulting ANNs with different noise levels. Incremental evolution is used where the task has different difficulty levels. The two methods compared are SANE with enforced sub-populations and delta-coding, and marker-based encoding, also an extension of delta-coding for marker-based encoding is suggested and compared. The hypothesis is that marker-based encoding is as capable even better on this problem as SANE with enforced sub-populations and delta-coding, and with delta-coding the computational power needed for marker-based encoding similar to SANE with enforced sub-populations and delta-coding.

In the dissertation it is shown that marker-based encoding is an efficient method for evolving the internal dynamics of robot controllers. Compared to other method tested it is much faster at evolving an agent capable of handling high level of difficulty with comparable complexity and robustness of the other evaluated methods. This method seems to be able to handle the aggressive reproduction, that deletes the bottom half of the population in the first generation, that is used in this dissertation (and in Gomez&Miikkulainen (1997)). This is interesting as in the implementations of marker-based encoding in Fullmer&Miikkulainen (1991) and Moriarity&Miikulainen (1995) the reproduction replaces only the bottom third of the population. Marker-based encoding is also shown to have an advantage in that the number of hidden nodes is under control of the evolution while SANE with enforced sub-populations and delta-coding requires the number of nodes to be predetermined which can cause difficulties as it is not possible to say what is the correct number of nodes for a specific problem. Marker-based encoding therefore spares the user the trial-and-error needed in SANE with enforced sub-populations and delta-coding.

On the other hand the hypothesis that delta-coding would speed up marker-based encoding turned out to be false. Two different versions were tested but they had no improving effects on evolutionary time. It has been shown that changing the conditions for creating new delta generation does not have the improving effects that was hoped for. This could be for a number of different reasons: the method used for deciding on a new delta step could be too simple to work, the aggressive reproduction used in this dissertation could better suited for marker-based encoding using only basic GA than on delta-coding.

6.7 Future work

An important future work is to create a better function for deciding when delta-coding is to create a new population. A good policy should make sure that the decrease in fitness that occurs when new population is created does not trigger another population change immediately, changing the method into inefficient random search. Delta-coding should then introduce new variation to the currently best solution on converging to a local maximum.

It would also be interesting to equip SANE with enforced sub-populations and delta-coding with one point arithmetic crossover, similar to the two point arithmetic crossover used in marker-based encoding with delta-coding. This would make the child produced be partially similar to each parent, now the child is simply made of mean values between the two parents with more bias to one of the parents.

Another important aspect to investigate further is to use large populations in the beginning of the evolution and decrease the number of individuals in a generation during evolution. This could counter the effects that the reproduction has by replacing the bottom half of the population in the first generation.

Interesting further work could also be to try using Cauchy distribution in the delta-coding of marker-based encoding to investigate whether the possibility of larger variations from current best solution improves the speed of evolution in marker-based encoding with delta-coding.

A comparison on the implications of using Gaussian distribution compared to Cauchy distribution in delta-coding is also an important future work, i.e. is it enough to increase the standard deviation to get as good results as using Cauchy distribution?

It is also interesting to see how marker-based encoding would perform on exactly the same task as Gomez&Miikkulainen (1997) used, the comparison would then be made on a simpler task that would show if marker-based encoding is better on both simple and complex robot tasks or just on more complex tasks.

Would it give better results to use 5 hidden nodes in SANE with enforced sub-populations and delta-coding, and use each hidden nodes as output as in Gomez&Miikkulainen (1997)? This would simplify the problem for SANE with enforced sub-populations and delta-coding in a way that cannot be done with marker-based encoding and could therefore show advantage over marker-based encoding in tasks that can be simplified.

Acknowledgements

I would like to thank the following:

Zoltán Biró, my supervisor, for his invaluable advises and help regarding my work. Björn Olsson for aqainting me with related work and helping in getting started on this dissertation. Faustino J. Gomez for his explanations of SANE with enforced sub-populations and delta-coding, and the task used in his work.

References

Braun, H. and Weisbrod, J. (1993). Evolving Neural Networks for Application Orianted Problems. In *Proc of the 2nd Ann Conf. On Evolutionary Programming*

Beer, R.D. (1995). A dynamic systems perspective on agent-environment interaction. *Artificial Intelligence* **72**: 173-215.

Cliff, D. and Miller, G.F. (1995). Tracking The Red Queen: Measurements of adaptive processing in co-evolutionary simulations. In *Advantages of Artificial Life: Proc. Of Third European Conf. On Artificial Life (ECAL95)*

Cliff, D. and Miller, G.F. (1996). Co-evolution of Pursuit and Evasion II: Simulation Methods and Results. In *proc. of the 4th International conf. On simulation of Adaptive Behaviour (SAB96)*

Elman, J. L. (1990). Finding Structure in Time. In *Cognitive Science*, **14**:179-211

Franklin, S. and Graesser, A. (1996). Is it an Agent or just a Program?: A Taxonomy for Autonomous Agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages.* Springer-Verlag

Fullmer, B. and Miikkulainen, R. (1991), Using Marker-Based Genetic Encoding Of Neural Networks To Evolve Finite-State Behavior. *In Proc. of The First Eropean Conference on Artificial Life (ECAL91)*

Gomez, F. and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior.* **5**:317-342

Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N., (1996a). Evolutionary Robotics: the Sussex Approach, In *Robotics and Autonomous Systems*. 1996 In Press

Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. (1996b). Evolutionary Robotics at Sussex, *In proc. of International Symposium on Robotics and Manufacturing (ISDRAM96)*

Hecht-Nielsen, R. (1990). *Neurocomputing*. Addison-Wesley Publishing Company, Inc. Reading, Massachusetts

Liu, Y. and Yao, X. (1996). A population-based learning algorithm which learns both architecture and weights of neural networks. In *Chinese Journal of Advanced Software Research*, **8(1)**:54-65

Mitchell, M. (1992). Genetic Algorithms, In *1992 Lectures in Complex Systems*. Nadel, L. and Stein, D. L. (editors). **1**:3-89

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts

Moriarty, D.E. and Miikkulaien, R. (1994). Evolving Neural Networks to focus minmax search. *In Proc. of the 12th National Conf. On Artificial Intelligence (AAAI-94)*

Moriarty, D. E. and Mikkulainen R. (1995). Discovering Complex Othello Strategies Through Evolutionary Neural Networks, *Connection Science*, **7(3)**:195-209

Moriarty, D. E. (1997). Symbiotic Evolution of Neural Networks in Sequential Decision Tasks, PhD thesis, Department of Computer Sciences, University of Texas, Austin, TX. Technical Report UT-AI97-259

Nolfi, S., Floreano D., Miglino, O., Mondada, F. (1994). How to evolve autonomous robots: different approaches in evolutionary robotics. In *Proceedings of the International Conference Artificial Life IV*, Cambridge MA: MIT Press

Russel, S. J., and Norvig, P. (1995). Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs, New Jersey

Steels, L. (1995). When are robots intelligent autonomus agents?. In *Journal of Robotics and Autonomous Systems*

Taylor, J. G. (1995). *Neural Networks*, Alfred Waller & Unicom, Oxon

Yao, X. and Liu, Y. (1997). A New Evolutionary System For Evolving Artificial Neural Networks, *IEEE Transactions on Neural Networks*, **8(3)**:694-713

Ziemke, T. (1997). The 'Environmental Puppeteer' Revisited: A Connectionist Perspective on 'Autonomy'. In *Proceedings of the 6th European Workshop on Learning Robots (EWLR-6), Brighton, UK*

Appendix A

In this appendix results from runs on improved delta-coding is presented. The results for SANE with enforced sub-populations and improved delta-coding are presented in A.1 and in A.2 the results for marker-based coding with improved delta-coding are presented.

Appendix A.1

This method also failed to produce a satisfactory results as it only made it to difficulty 4 in 10000 generations. This deems the method as a failure. Figure 14 shows the fitness values of the evolution of a representative run. The vertical dotted lines show when difficulty level increases. The difficulty levels are marked with "d:*n*" in the figure, where *n* is the difficulty level.

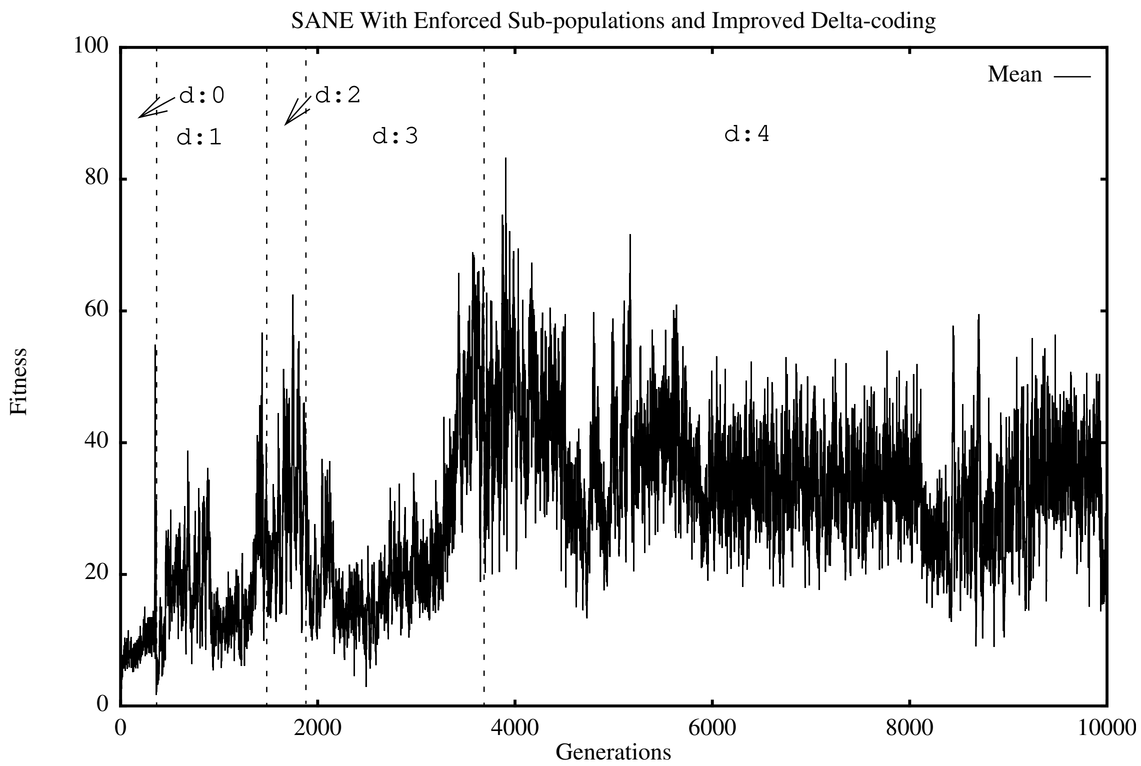


Figure 14: Fitness values of generations in SANE with enforced sub-populations and delta-coding

As this method also failed to get to difficulty level 7 tests with noise are not run.

The generations for each difficulty level is shown in table 8 below.

TABLE 8.

Generation	Difficulty
0 - 351	0
351 - 1435	1
1435 - 1748	2
1748 - 3906	3
3906 - 10000	4

Appendix A.2

This method failed to increase the speed of evolution of marker-based encoding with delta-coding. Figure 14 shows the fitness values of the evolution of a representative run. The vertical dotted lines show when difficulty level increases. The difficulty levels are marked with "d:*n*" in the figure, where *n* is the difficulty level.

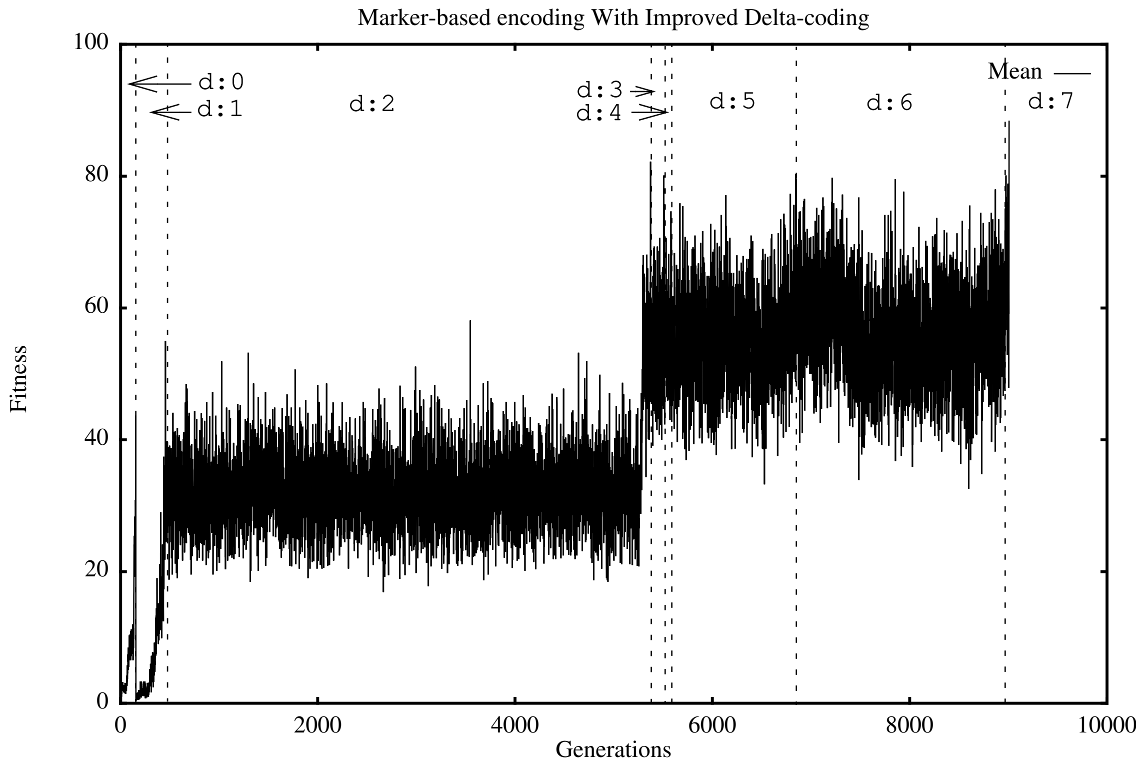


Figure 15: Fitness values of generations in marker-based encoding with improved delta-coding

As this method failed to improve the speed of marker-based encoding tests with noise are not run.

The generations for each difficulty level is shown in table 9 below.

TABLE 9.

Generation	Difficulty
0 - 155	0
155 - 457	1
457 - 5295	2
5295 - 5374	3
5374 - 5509	4
5509 - 6847	5
6847 - 8976	6
8976 - 9008	7

Appendix B

In this appendix the results from runs on SANE with enforced sub-populations and delta-coding with additional nodes and standard deviation 6 are presented. In appendix B.1 the results with 18 hidden nodes and standard deviation 1 are shown. In appendix B.2 the results with 16 hidden nodes and standard deviation 6 are shown and in appendix B.3 the results with 18 hidden nodes and standard deviation 6 are shown. The results with 16 hidden nodes and standard deviation 1 are not included as it did not get passed difficulty 1. Due to lack of time it was not possible to test the noise tolereance of the evolved ANNs.

Appendix B.1

This run of SANE with enforced sub-populations and delta-coding using 18 hidden nodes and standard deviation 1, produced satisfactory results in 4662 generations. Figure 16 the fitness values of the evolution of a representative run. The vertical dotted lines show when difficulty is increased. The difficulty levels are marked with "d:*n*", where *n* is the difficulty level.

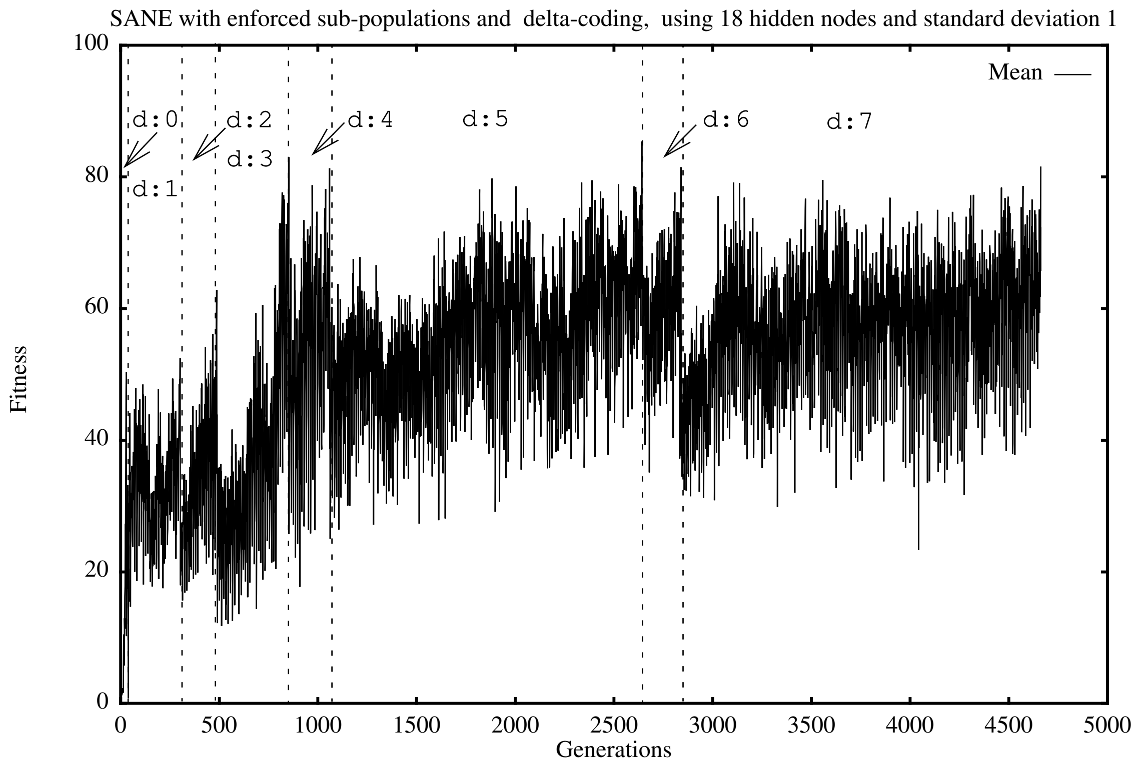


Figure 16: SANE with enforced sub-populations and delta-coding, using 18 hidden nodes and standard deviation 1

The generations for each difficulty is shown in table 10 below.

TABLE 10.

Generation	Difficulty
0 - 29	0
29 - 302	1
302 - 487	2
487 - 852	3
852 - 1059	4
1059 - 2641	5
2641 - 2839	6
2839 - 4662	7

Appendix B.2

This run of SANE with enforced sub-populations and delta-coding using 16 hidden nodes and standard deviation 6, produced satisfactory results in 2790 generations. Figure 17 the fitness values of the evolution of a representative run. The vertical dotted lines show when difficulty is increased. The difficulty levels are marked with "d:*n*", where *n* is the difficulty level.

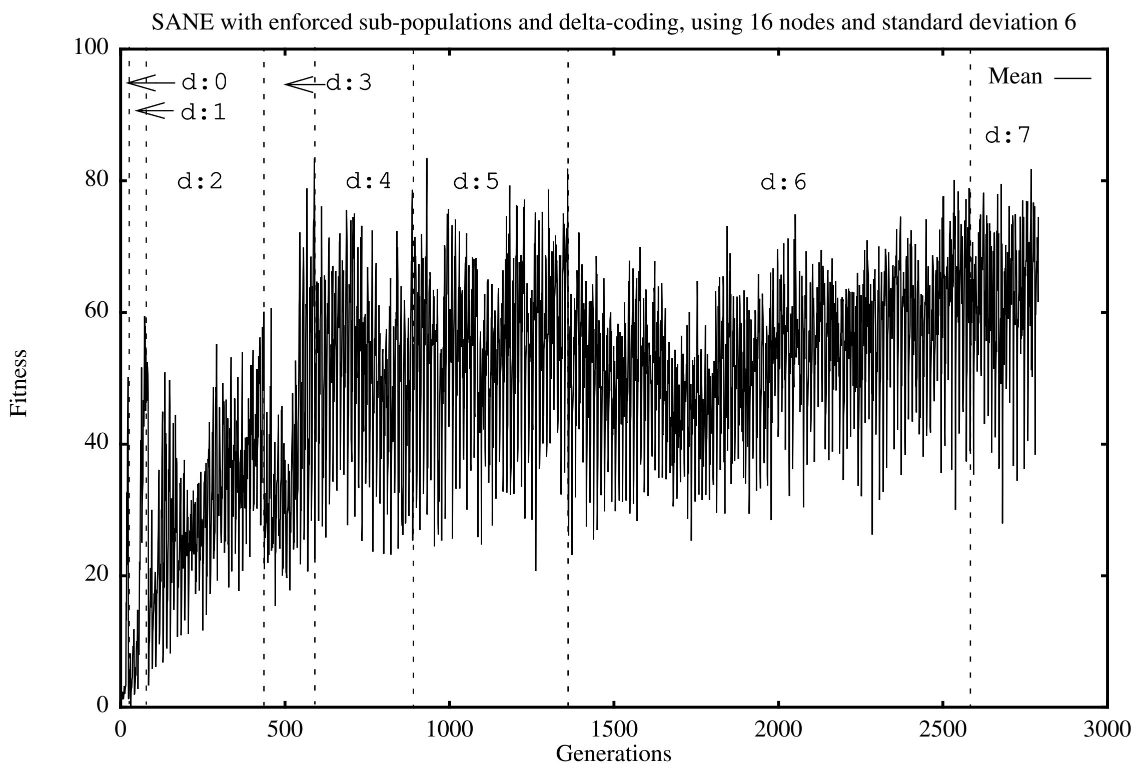


Figure 17: Fitness values of generations in SANE with enforced sub-populations with delta-coding, using 16 nodes and standard deviation 6

The generations for each difficulty is shown in table 11 below.

TABLE 11.

Generation	Difficulty
0 - 23	0
23 - 64	1
64 - 436	2
436 - 589	3
589 - 931	4
931 - 1360	5
1360 - 2534	6
2534 - 2790	7

Appendix B.3

This run of SANE with enforced sub-populations and delta-coding using 18 hidden nodes and standard deviation 6, produced satisfactory results in 1787 generations. Figure 18 the fitness values of the evolution of a representative run. The vertical dotted lines show when difficulty is increased. The difficulty levels are marked with "d:*n*", where *n* is the difficulty level.

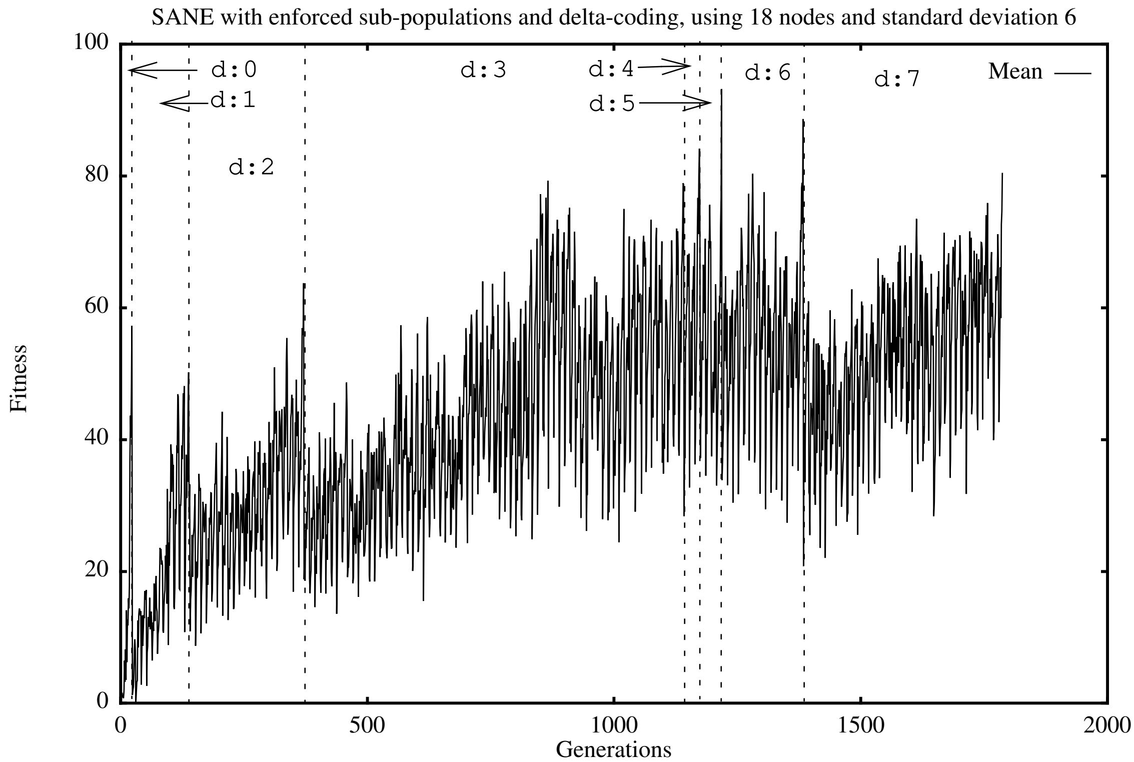


Figure 18: Fitness values of generations in SANE with enforced sub-populations with delta-coding, using 18 nodes and standard deviation 6

Appendix B

The generations for each difficulty is shown in table 12 below.

TABLE 12.

Generation	Difficulty
0 - 23	0
23 - 138	1
138 - 371	2
371 - 1173	3
1173 - 1218	4
1218 - 1281	5
1281 - 1383	6
1383 - 1787	7