Evolution of An Autonomous Learning Agent

JAKOB LUBLIN

KTH Computer Science and Communication

Master of Science Thesis
Stockholm, Sweden 2006
Evolution of An Autonomous Learning Agent

J A K O B L U B L I N

Master’s Thesis in Computer Science (20 credits) at the School of Electrical Engineering Royal Institute of Technology year 2006
Supervisor at CSC was Örjan Ekeberg
Examiner was Anders Lansner

TRITA-CSC-E 2006:009
ISRN-KTH/CSC/E--06/009--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.csc.kth.se
Abstract

This thesis describes the development of a tool based on an evolutionary algorithm, its purpose is to optimize and analyze the parameters of an autonomous software agent. The objective of the agent is to solve a natural resource management problem, i.e. to find the optimal harvest rate of a fish population.

The behavior of a single agent when managing resources modeled by two different ecology functions is analyzed. The ecology functions are based on logistic growth with a constant effort harvest function and logistic growth with a Monod harvest function. The later presents the agent with a resource that will collapse if it is excessively exploited.

Four different evolutionary strategies are evaluated for the optimization. Two of them utilize recombination the other two do not. The strategies without recombination have the best performance with both of the ecology functions and delivers well suited parameter settings for the agent.

In this thesis the objective of the agent has been to find the optimal harvest rate of the population. However, the fitness function of the ES might easily be modified which will allow the agent to be optimized for other behaviors, e.g. to make the agent simulate human behavior.
Evolution av en självlärande agent

Sammanfattning


Fyra olika typer av evolutionära algoritmer har utvärderats. Två av dem använder rekombination, vilket dom andra två inte gör. De algoritmer som inte använder rekombination uppförde bäst prestanda och genererade bra parametervärdet till agenten.

Målet med det här examensarbetet har varit att hitta de parameterinställningarna som maximerar agentens långsiktiga fångst. Men verktyget går enkelt att anpassa för att hitta parameterinställningar som får agenten att simulera andra typer av beteende, t.ex. ett mer kortsiktigt mänskligt beteende.
Contents

1 Introduction ........................................... 1
   1.1 An Agent ........................................... 1
   1.2 Reinforcement Learning ............................. 2
   1.3 Artificial Neural Networks .......................... 4
   1.4 Evolutionary Algorithms ............................ 5
   1.5 The Objective ....................................... 6

2 Theory .................................................. 8
   2.1 Ecological Systems ................................. 8
   2.2 Temporal-Difference ................................ 12
   2.3 Radial Basis Networks .............................. 14
       2.3.1 Network training ............................. 15
   2.4 Evolutionary Strategies ............................ 16
       2.4.1 Recombination ............................... 17
       2.4.2 Self-adaption ............................... 18
       2.4.3 Noise ....................................... 21

3 Methods ............................................... 23
   3.1 Object Parameters .................................. 23
   3.2 An ES Algorithm ................................... 24
       3.2.1 ES Parameters ............................... 25
       3.2.2 Recombination ............................... 25
       3.2.3 Fitness Functions ............................ 26
   3.3 A Brute Force Algorithm ........................... 27

4 Results ............................................... 28

5 Discussion ............................................ 35
   5.1 Constant Effort Harvest ............................ 35
   5.2 Monod Harvest Function ............................. 35

6 Conclusions ......................................... 38

References .............................................. 39
1 Introduction

Understanding how to set human society on track toward a sustainable management of its life-support systems is a complex challenge. Solving the problem involve the understanding of ecosystem functioning, the human actors, their learning processes and the governance of the actors.

Jon Norberg has developed methods to use social network theory in order to study natural resource management problems [19]. The methods include an agent-based framework for studying learning and information sharing processes, i.e. the role of different informational network structures, among agents that interact with simulated “natural” systems.

The representation of human learning is a vast research field and any mental model can only capture part of human learning dynamics. Hence three constraints are imposed on the agents to capture some parts of human learning. First, the agent can never attain a complete understanding of the processes underlying the natural resource. Second, the agent might only learn by experimentation and information sharing. Third, the memory of the agent is limited.

This thesis deals only with individual agents. I will try to develop a tool that will assist in the pursuit of desirable parameter values and that helps in the analysis of how the behavior of the agent is affected by different parameter settings.

The agent studied in this thesis utilizes methods from the machine learning field to solve a natural resource management problem, i.e. to optimally control the harvest rate of a fish population.

1.1 An Agent

A software agent is an autonomous program that can perceive its environment through sensors and influence it by actuators, i.e. an agent has some method of sensing the environment or getting input from it and has the ability to take actions to influence it. The Agent must not be controlled by an user, an other agent or anything else, it must act on itself.

The Agent developed by Jon Norberg and Emilie Lindkvist utilizes reinforcement learning and radial basis networks. It is written in JAVA and can be called from Matlab. The aim of the agent is to solve a natural resource management problem, i.e. how to regulate the harvest of a fish population so that maximum return is attained.

Reinforcement learning is a group of machine learning algorithms inspired by basic learning methods observed in humans and animals. It is a reinforcement learning algorithm that controls the actions of the agent. Radial basis
networks are a class of artificial neural networks (ANN), they are inspired by biological neuron networks such as the brain. ANNs are often used for classification, pattern recognition and function approximation. The reinforcement learning algorithm attempts to represent its environment by a function, this function is approximated by a RBF (radial basis network).

The objective of the agent is to maximize the long-term harvest of a fish population. The state of the population and the present harvest constitute as the input, or the environment, and the agent control the effort of the harvest being imposed on the population.

If the effort is increased too much the immediate harvest will rise, but the long-term harvest will decline. To handle this, the agent has to consider not only the immediate reward of an action, but also the effect that action has on future rewards. Reinforcement learning methods are good at handling this problem.

1.2 Reinforcement Learning

Learning in its most basic form is achieved by an agent interacting with its environment and then evaluating the result of that interaction. If the agent finds the result favorable, it will reinforce the association between the previous state of the environment and that action. If the result is poor, the association will be weakened. In RL terminology reward is used to designate the response from the environment. A high reward will strengthen the association between action and state.

This method is quite effective, however in many problem situations, such as the one discussed before, it will fail. If the agent only takes into account its immediate reward, it will set the harvest rate too high. Given an arbitrary population, the more fish caught the higher the immediate reward. The absolutely highest reward will be received if the entire population is caught at once. This will result in the extinction of the population.

To solve this problem RL methods assign a value to each state. The value is based on the reward received and the value of the next state. When an agent takes an action it receives a reward and it ends up in a new state. One simple method to update the old state is to take the sum of the reward and the value of the new state as the update.

Reinforcement learning is not defined by a method of solving a problem, instead it is a specific problem that is used as the definition. A formal description of the specific problem is to optimally control a Markov Decision Process [27], and any algorithm that solves this problem is said to be a reinforcement learning algorithm. A more trivial definition states that a RL agent is an agent interacting with its environment to achieve a goal.
1.2 Reinforcement Learning

The field of reinforcement learning [16] unites many different disciplines, from psychology and neuroscience to statistics and computer science and the methods have been tested with success on many different real world problems [25, 9, 18].

There are three fundamental classes of reinforcement learning methods: Dynamic Programming, Monte Carlo, and Temporal Difference. These classes share the same three main sub elements, they have a policy function, a reward function and a value function. They might also have a model of the environment.

The policy function maps states to actions, it tells the agent which action to take in a particular state. An important concept in machine learning is the balance between using current knowledge and to obtain new. This is called the trade-off between exploitation and exploration, and the policy function determines this trade-off. The policy might be greedy, always selecting the best action that is known at the moment. However, a greedy policy inhibits learning and is not good in the long-term. To learn new information about its environment the agent sometimes has to explore and make actions that, at the moment, is thought to be poor.

The reward is the signal from the environment that tells the agent how good a state is. The signal is generated by the reward function which is actually considered to be a part of environment. This function is very important because it determines what the agent actually learns. The agent attempts to maximize the total rewards and if this function is poorly constructed the agent will not learn its environment correctly.

The purpose of the value-function is to assign a value to each state. This value is an estimate of the rewards that the agent will receive if it passes trough that state and takes actions according to its policy function. Instead of a value function many reinforcement learning methods use an action-value function that map values to action-state pairs, this function is called a Q-function.

Dynamic Programing [2] is the most mathematically developed class of methods that solves the reinforcement learning problem, they are off-line bootstrapping algorithms that need accurate and complete models of their environment.

An off-line algorithm needs to complete an episode before it can update its value and policy functions, therefore it can not learn while it explores. For example when an off-line agent learns how to play chess, it has to wait until the game is over before it can update its policy and value functions.

Bootstrapping is a term used to describe how the update is done. When a new value is being calculated in a bootstrapping algorithm, the value of the next state and the reward received is used. A non bootstrapping algorithm
uses the total reward received during the episode to calculate the new values.

Monte Carlo Methods [21, 3] do not require a perfect model of the environment, they can be used in both on-line and off-line training tasks but the task must be of an episodic character because Monte Carlo methods do not bootstrap. They must know the total reward for the episode before they update the value function.

Temporal Difference Methods [26] can be thought of as a combination of Dynamic Programming and Monte Carlo Methods, they do not need a model of the environment and they bootstrap. This makes them suitable for many types of problems especially on-line problems. The agent, in the current study, utilizes a Temporal Difference Method named Watkins Q-learning.

Sutton [26] has shown that temporal difference methods require less memory and peak computational power and still produce more accurate predictions then the other classes of RL methods. Samuel [22] studied TD methods as early as 1959 using the game of checkers, Tesauro successfully constructed a backgammon playing program [28, 29] and lots of other successful TD studies [4, 31] have been performed.

1.3 Artificial Neural Networks

Artificial neural networks (ANN) are algorithms inspired by the nervous system of biological organisms. They are used for many different applications, for example pattern recognition, classification, noise reduction and function approximation.

A neural network consists of individual neurons that are interconnected. The most common network layout is the feed forward network. In this architecture the neurons are ordered in layers and individual neurons are only connected with neurons in adjacent layers. The information flows from the input layer through the hidden layers to the output layer.

The following equation [17] describes the output of a single neuron:

\[ y = f\left(\sum_{i=1}^{n} x_i w_i - \theta_i\right) \]

where \( f(\cdot) \) is the activation function, \( x_i \) the input from node \( i \) of the previous layer, \( w_i \) a weight parameter and \( \theta_i \) a threshold parameter.

A biological neuron fires, or becomes active, when the weighted sum of the inputs reaches a threshold value, this is simulated by the activation function and the most commonly used activation function is the sigmoid function.

The memory of an ANN is stored in the layer weights and it is those that are updated when the network is trained.
1.4 Evolutionary Algorithms

Survival of the fittest is the driving force behind natural evolution. Those individuals that are best adapted for the environment will have the best chance of survival and producing offspring. The genes of these individuals will make up an increasing part of the gene pool for every new generation. The result of this process is that every new generation is better adapted to its environment than the previous one.

J. H. Holland was inspired by Darwin and Russels theories of evolution and started to develop a class of optimization algorithms called Genetic Algorithms [15]. These algorithms use a set of binary strings, where each string represents a valid solution to the problem being optimized. The entire set is called a population and the solutions are called individuals.

In the beginning of an optimization process a random population is generated. Each solution is then evaluated and has a fitness value assigned that depends on how well it performed. When every solution has a fitness value assigned to it, the selection phase begins.

The selection method can vary but a common method is stochastic sampling with replacement [13], also known as proportional selection.

\[
p_i = \frac{f_i}{\sum_{i=1}^{N} f_i}
\]  

(1)

This method assigns a probability to each solution according to equation (1). Using this probability distribution individuals are selected for further manipulation.

The next standard manipulation method is the crossover operation. Two individuals are selected randomly from the population. The strings that make up the individuals are then split, at a point that can be fixed or vary randomly. These four substrings are then joined so that two new strings are formed. Not every individual of the population is selected for the crossover operation, the number is governed by a crossover probability.

When the crossover operation is finished the mutation operation begins. Every bit of every individual will have a very small probability of mutating, which means that its value will be inverted. The role of the mutation operation is to guard against sets of solutions being prematurely lost by the crossover operation. It is the crossover operation that is the main search method for a GA.

Holland described a forth genetic operation, the inversion operation. This operation selects two points on a string randomly, it then inverts the values
of the bits between those points. This operation has been observed in the
nature but is not commonly used by genetic algorithms.

Evolution strategies are another class of evolutionary algorithms, they
were initially developed for the purpose of parameter optimization. One of
the initial developers Rechenberg once said [24]:

"the method of organic evolution represents an optimal strategy
for the adaption of living things to their environment... [and] ...
  it should therefore be worthwhile to take over the principles of
  biological evolution for the optimization of technical systems".

Unlike genetic algorithms, evolution strategies do not make use of binary
strings to encode the solutions. Instead they use the parameter values di-
rectly. Another dissimilarity to GA's is that ES uses the mutation operation
as its main search operation.

Evolutionary algorithms may appear random, but they efficiently use
historical knowledge to select new search points with expected improvement
and they are not limited to assumptions about the search space, assumptions
like continuity and the existence of derivatives.

1.5 The Objective

The purpose of this thesis is to develop a Matlab tool that will aid in the
pursuit of suitable values for the parameters of the agent. There are many
parameters that governs the behavior of the agent. In this thesis only four
of them are investigated, i.e. the number of radial basis functions, the step
size of the RBF training algorithm, the step size of the TD update and the
temperature of the softmax action selection.

This thesis only deals with individual agents, acting on themselves. The
agents tries to solve a small-scale natural resource management problem.
More specifically they try to find the optimal harvest effort for a fish popu-
lation.

Two different ecological models are investigated in this thesis. Both of the
models are based on logistic growth but they use different harvest functions.
The first ecological model to be investigated utilizes a constant effort harvest
function and the second utilizes a Monod harvest function. Both of these
harvest functions have a parameter that represents the effort exerted by the
harvester of the population. This parameter is controlled by the agent and
constitutes as the agents action, i.e. this parameter is the element of the
environment that the agent is able to influence.

This thesis will try to optimize the behavior of the agent toward the
maximum long term harvest of a population. However, sometimes other
behavior is desired, for instance the imitation of real human behavior. The tool should be able to find values for the parameters so that the agent might be used to simulate other behavior than the optimal.
2 Theory

In this section the theory behind Evolution Strategies, Temporal-Difference, Radial Basis Networks and ecological system are examined.

Evolution Strategies are used to obtain suitable parameters for the Temporal-Difference algorithm and Radial Basis Network of the agent and the main focus of this section is on the theory of Evolution Strategies.

Temporal-Difference is the class of Reinforcement Learning algorithms which drive the learning of the agent. The Radial basis network is used to approximate internal functions of the TD algorithm.

The task of the agent studied in this thesis is the management of a natural resource, specifically the optimal harvest of a fish population. Mathematical models of the population growth and the harvest of a population are examined in the ecological system subsection.

2.1 Ecological Systems

A simple mathematical model of the population growth is the Malthus’ law from 1798 [8]. It arises from the solution of the initial value problem,

\[
\frac{dx}{dt} = rx, \quad x(0) = x_0
\]  

where \( r = b - \mu \) denotes the constant growth rate per capita, \( b \) is the average birth rate and \( \mu \) is the average death rate. If \( r \) is positive, equation (2) will lead to a population with exponential growth and if \( r \) is negative the equation will lead to extinction.

Malthus law might be used to describe the growth rate of a population living in an environment with almost unlimited resources relative to the size of the population. This is the case in the beginning of an epidemic.

Most environments do not provide unlimited resources and can therefore not support exponential growth very long. This can be remedied if the constant \( r \) is replaced by a function \( G \) that is dependent of the size of the population. The new equation then becomes:

\[
\frac{dx}{dt} = xG(x), \quad x(0) = x_0.
\]  

The most common example of \( G(x) \) is Verhulst from 1838,

\[
G(x) = r(1 - \frac{x}{K}),
\]  

where \( K \) is the carrying capacity of the environment. This equation is known as the logistic growth equation. It describes the growth of a population when there is a limit to the resources available.

\section{Applications of Evolution Strategies}

Evolution Strategies have been applied in various fields such as optimization, control, and machine learning. They are particularly useful when dealing with complex, high-dimensional problems where traditional optimization methods may struggle.

\subsection{Evolution Strategies in Optimization}

Evolution Strategies have been successfully applied to a wide range of optimization problems, including function optimization, machine learning, and parameter tuning. They are particularly effective in problems where the objective function is non-convex, multimodal, or noisy.

\subsection{Evolution Strategies in Control}

Evolution Strategies have been used in the design of control systems, particularly in the context of reinforcement learning. They have been applied to problems such as the control of robotic systems, autonomous vehicles, and other complex systems.

\subsection{Evolution Strategies in Machine Learning}

Evolution Strategies have been applied in machine learning to optimize hyperparameters of models, select features, and improve model performance. They are particularly useful in scenarios where the model architecture is fixed, but the hyperparameters need to be optimized.

\section{Radial Basis Networks}

Radial Basis Networks (RBNs) are a type of artificial neural network that uses radial basis functions as activation functions. They are known for their ability to approximate any continuous function, making them suitable for a wide range of applications.

\subsection{Radial Basis Functions}

Radial Basis Functions (RBFs) are functions that depend only on the distance from a central point. They are used as the basis functions in RBNs and are known for their ability to provide a smooth interpolation between training data points.

\subsection{Training RBNs}

Training RBNs involves determining the values of the weights and the centers of the RBFs. This is typically done using an iterative optimization algorithm, such as the conjugate gradient method or the Levenberg-Marquardt algorithm.

\section{Conclusion}

In conclusion, Evolution Strategies and Radial Basis Networks are powerful tools that can be applied to a wide range of problems. They have the potential to revolutionize the way we approach complex, high-dimensional problems in fields such as optimization, control, and machine learning. Further research into these areas is likely to lead to significant advancements in the field.
which combined with equation (3) results in the logistic equation [8]:

\[
\frac{dx}{dt} = rx\left(1 - \frac{x}{K}\right).
\]  
(5)

The constant \( K \) is called the carrying capacity and is the maximum size of the population that the environment can support.

When the population begins to reach the carrying capacity, the growth rate will decline and eventually drop to zero. And when the size of the population becomes very small the logistic growth model can be approximated by Malthus law of exponential growth.

\[
x \rightarrow K \Rightarrow \frac{dx}{dt} \rightarrow 0
\]

\[
x \rightarrow 0 \Rightarrow \frac{dx}{dt} \approx rx
\]

The maximum growth rate is called the maximum sustainable yield. To find it derivate equation (5) with respect to \( x \), and set the derivate to zero. For the logistic function the maximum sustainable yield is found to be at half the carrying capacity, \( x_{msy} = K/2 \).

\[
\left\{ \begin{array}{l}
f'(x) = r - \frac{2 rx}{K} \\
f'(x) = 0
\end{array} \right. \Rightarrow x = \frac{K}{2}
\]

A population modeled by a differential equation

\[
x' = f(x)
\]

is being harvested, it can then be described by:

\[
x' = f(x) - h(x)
\]

where \( h(x) \) is a function describing the rate of the harvest.

If the harvest function is constant, \( h(x) = H \), the harvest is called constant yield harvesting [8]. This means that a constant number of members of the population are being caught every time interval.

An example of a population being subjected to this kind of harvest is the Swedish elk population, where hunters are allowed to kill a constant number of elks each season. The elks natural predators are too few to make any significant impact on the population.

Populations exhibiting logistic growth being subjected to constant yield harvesting can be described by the following equation:

\[
\frac{dx}{dt} = rx\left(1 - \frac{x}{K}\right) - H.
\]  
(6)
If the harvest rate is less than the current growth rate, the size of the population will stabilize. There exists two equilibrium points, they are located where the constant line $h(x) = H$ intersects the logistic growth curve, see figure 1 on page 12. The first equilibrium is unstable, so the size of the population will not stabilize there, the second however is stable.

If the size of the population is not large enough to generate the growth required to sustain the harvest, it will crash and become extinct. This will happen if the size of the population is to the left of line $A$ in figure 1 (b).

$$H < \text{MSY} \Rightarrow x \rightarrow 0$$
$$H = \text{MSY} \Rightarrow x \rightarrow x_{\text{msy}}$$
$$H > \text{MSY} \Rightarrow x \rightarrow 0$$

Another kind of harvest is the constant effort harvest [8], which assumes that the harvest is proportional to the size of the population and the effort expended. The harvest function becomes a linear function, $h(x) = Ex$. $E$ is the effort expended, this can for example be the number of boats in a fishing fleet or the total time trolling. The logistic equation with constant effort harvest becomes:

$$\frac{dx}{dt} = rx\left(1 - \frac{x}{K}\right) - Ex. \quad (7)$$

This function has two equilibrium points, $x_{\infty}^1$ and $x_{\infty}^2$, shown below. The first equilibrium $x_{\infty}^1$ is unstable while $x_{\infty}^2$ is stable.

$$\frac{dx}{dt} = 0 \Rightarrow \begin{cases} 
  x_{\infty}^1 = 0 \\
  x_{\infty}^2 = \frac{K(r-E)}{r} \text{ if } 0 \leq E \leq r
\end{cases}$$

The yield of the harvest at equilibrium is given by:

$$h_{x_{\infty}} = E \frac{K(r-E)}{r}, \quad (8)$$

where $x_{\infty}$ is the size of the population at equilibrium. To find the value of $E$ which gives the maximum harvest rate, derivate $h_{x_{\infty}}$ with respect to $E$ and set the derivate to zero:

$$\begin{cases} 
  h'_{x_{\infty}} = K(1 - \frac{2E}{r}) \\
  h'_{x_{\infty}} = 0
\end{cases} \Rightarrow E_{\text{max}} = \frac{r}{2}.$$

Combining $E_{\text{max}}$ with equation (8) we obtain the maximum harvest rate of the constant effort harvesting:

$$h_{\text{max}} = E_{\text{max}} \frac{K(r-E_{\text{max}})}{r} = \frac{rk}{4}.$$
The equilibrium $x_\infty$ can also be found where the line $h(t) = Ex$ intersects the logistic growth curve, see figure 1 on the following page.

A third harvest function to be discussed is the Monod or Hollings type II function [10]:

$$h(x) = E \frac{x}{k + x}$$

where $k$ is the half saturation constant, at this value the harvest rate is half of its maximum.

When the size of the population is small the Monod harvest function behaves similar to the constant effort harvest function, but as the size of the population increases its behavior gradually becomes more like the constant yield harvest function.

The Monod harvesting function might be used to describe a fishing fleet, where $E$ is a constant describing the size of the fleet and the time spent fishing. When the size of the fish population is relatively small the catch is almost proportional toward the time spent fishing. However, as the size of the population grows larger, factors such as the size of the cargo-holds of the fishing vessels and the capacity to process the caught fish begin to limit the harvest rate. As the size of the population increases the harvest rate becomes more similar to constant yield harvest.

Logistic growth subjected to a Monod harvest function becomes:

$$\frac{dx}{dt} = rx(1 - \frac{x}{K}) - E \frac{x}{k + x}. \quad (9)$$

The behavior of equation (9) will be described by examining three different Monod harvest functions shown in figure 1 (a). If the population is being harvested according to line I, the harvest rate is always larger then the growth rate, this will inevitable lead to a population crash.

The second scenario (line II) has similar characteristics as constant yield harvest. The logistic curve and the harvest curve intersect at the vertical line $A$. To the left of this intersection the harvest rate is larger than the growth rate, which results in a crash. However if the size of the population is to the right of this intersection, where the growth is larger than the harvest, it will stabilize at the equilibrium point where the second intersection occurs (line $B$).

The third scenario is similar to constant effort harvesting. The size of the population will stabilize at the equilibrium point marked by line $C$. 

2.1 Ecological Systems
(a) Three different Monod harvest functions. (b) Three different harvest functions.

Figure 1. a) The Monod harvest function plotted with three different values of the parameter E. b) Logistic growth plotted against maximum yield harvest, maximum effort harvest and the Monod harvest function.

2.2 Temporal-Difference

The most basic TD method is the \( TD(0) \) [27], the update of its value function is shown below:

\[
V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]
\]

where \( V \) is the value function, \( s_t \) is the state at time \( t \), \( \alpha \) is the step-size, \( \gamma \) is the discount-rate and \( r_{t+1} \) is the reward received at time step \( t + 1 \).

TD methods are on-line bootstrapping methods, which means that the update is performed every time-step and is based on an estimate. The estimate consist of the value of the next state, \( V(s_{t+1}) \), which is also an estimate and the reward received during the state transition, \( r_{t+1} \).

The step-size parameter \( \alpha \) influences the agents learning rate, a large value makes the method learn fast, but will make it sensitive to noise. If the step-size is too large the value function might begin to oscillate around the correct value and if it is too small the method will learn the value function too slowly.

Some methods change the step-size over time, usually starting with a large value and then gradually lowering it as the methods approximation of the value function gets better.

The discount-rate \( \gamma \) influences the evaluation of future rewards. A small discount-rate will make the method shortsighted, it will try to maximize
2 THEORY

2.2 Temporal-Difference

| Initialize $V(s)$ or $Q(s, a)$ arbitrarily |
| Repeat (for each episode) |
| Initialize $s$ |
| Repeat (for each step of the episode): |
| $a \leftarrow$ action given by $\pi$ or derived from $Q$ |
| Take action $a$; observe reward $r$ and next state $s'$ |
| $V(s) \leftarrow V(s) + \alpha [r + \gamma V(s') - V(s)]$ $\text{TD}(0)$ |
| $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, s)]$ $\text{Q-learning}$ |
| $s \leftarrow s'$; |
| until $s$ is terminal |

**Figure 2.** Procedural form of the $\text{TD}(\theta)$ or Watkins $\text{Q}$-learning method.

its immediate reward without consideration that its actions might lead to smaller rewards in the future.

$\text{TD}(0)$ methods follow a separate policy function, $\pi(s)$, when deciding which action to take. The policy function might be fixed or be updated similarly to the value function. Another method is to construct an action-value function, a $\text{Q}$-function, and derive the policy from that function.

Watkins $\text{Q}$-learning [30] uses a $\text{Q}$-function, its update is on the form:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, s_t)].$$ (11)

It is an off-policy method which means that it does not follow the policy when making the update only when selecting an action.

Policies used by $\text{TD}$-methods can differ considerably. Three different policies will be described; greedy, $\epsilon$-greedy and softmax.

The greedy policy selects the action that maximizes the $\text{Q}$-function, $\max_a Q(s, a)$. The $\epsilon$-greedy policy is similar to the greedy policy, the difference is that it sometimes selects a totally random action instead of the best one. The probability of this happening is set by the variable $\epsilon$.

Softmax [27] is a more advanced policy, it uses a Boltzmann probability distribution shown below:

$$\frac{\exp(Q_t(s, a) / \tau)}{\sum_b^{N} \exp(Q_t(s, b) / \tau)}$$ (12)

to assign a probability to each action. The better an action is thought to be the higher the probability of it being selected. The parameter $\tau$ is called the temperature, it influences the exploitation-exploration trade off. A low temperature makes the policy selection greedy and a high makes it more explorative.
2.3 Radial Basis Networks

Radial basis functions networks (RBFs) are a class of multi layered neural networks. They are often used for classification problems and function approximation. The Q-function, part of the reinforcement learning algorithm, is approximated by a RBF.

The network consists of an input layer, a hidden layer and an output layer. The radial basis functions are located in the nodes of the hidden layer. A radial basis function network is described by the following equation [7]:

\[ \hat{f}_k(\vec{x}) = \sum_{j=1}^{M} w_{kj} \phi_j(\vec{x}), \quad \text{were } \vec{x} = \{x_1, x_2, \ldots, x_d\} \quad (13) \]

where \( f_k \) is the output of node \( k \), \( \vec{x} \) is the input vector, \( w_{kj} \) the layer weight matrix and \( \phi_j(\cdot) \) the transfer function. The transfer function consists of a number of radial basis functions and the layer weight matrix is used for the weighted summation of their output before the output layer of the network.

The most commonly used basis function is a gaussian shown below in equation (14) on the following page, however many other functions can be
used, for instance spline-, multi-quadric- and linear-functions [7].

\[
\phi_j(\vec{x}) = \exp \left(-\frac{||\vec{x} - \vec{\mu}_j||^2}{2\sigma_j^2}\right), \quad \text{were} \quad \vec{\mu}_j = \{\mu_{j,1}, \mu_{j,d}, \ldots, \mu_{j,d}\} \quad (14)
\]

The vector \(\vec{\mu}_i\) designates the center of the basis function and the function is radially symmetric around this point, hence the name radial basis function. The parameter \(\sigma\) controls the size of the gaussian bell and the smoothness of the output function.

The agent uses a variant of equation (14) shown below,

\[
\phi_j(\vec{x}) = \exp \left(-\frac{||\vec{x} - \vec{\mu}_j||^2}{2\sigma_j^2}\right), \quad \text{were} \quad \vec{\mu}_j = \{\mu_{j,1}, \mu_{j,d}, \ldots, \mu_{j,d}\} \quad (15)
\]

where the parameter \(\sigma_j\) is changed to a vector \(\vec{\sigma}_j\). This change allows for different variances of the basis function in different dimensions. This is important because of the different ranges of the action and state spaces.

The problem being dealt with in this thesis specifies the number of nodes in the input and the output layer. There are two nodes in the input layer, one for the action and one for the state. The output layer consists of one node, since every action-state pair is mapped to one Q-value.

### 2.3.1 Network training

The training of a radial basis network is usually a two-stage process, where the parameters of the basis functions are updated in the first stage and the
layer weight matrix $w_{kj}$ in the second. However, when the agent is created the
positions of the basis functions, $\bar{\mu}_j$, are randomly distributed and the variance
vectors, $\sigma_j$, are set based on the ranges for the different input variables. These
values are not changed after the initiation, which makes the first stage of the
training process unnecessary.

The second stage uses the delta rule, a gradient descent method, to update
the layer weight matrix. First an error function is calculated according to:

$$E = \sum_{k=1}^{n} (\hat{f}_k(\bar{x}) - f_k(x))^2,$$

(16)

where $\hat{f}_k$ is the output from the network and $f_k$ is the function value given
by the RL algorithm.

When the error function has been calculated the weight matrix is updated
according to [17]:

$$w_{jk}(t + 1) = w_{jk}(t) + \delta \frac{\partial E}{\partial w_{jk}(t)}$$

(17)

where $\delta$ is the stepsize.

2.4 Evolutionary Strategies

The simplest and most basic ES is the $(1+1)$ algorithm. One parent generates
one child through mutation, they are evaluated and the fittest of the two
survive to become the parent of the next generation. A solution is represented
by a parameter vector. The mutation is achieved by random numbers being
added to the parameters according to equation (18),

$$x_i' = x_i + N_{0,\sigma_i}$$

(18)

where $N_{0,\sigma_i}$ is a gaussian distributed random number with zero mean and
standard deviation $\sigma_i$.

Rechenberg used this algorithm to calculate the optimal mutation rate
and came up with his $1/5$ success rule [20]:

"The ratio of successful mutations to all mutations should be $1/5$.
If it is greater than $1/5$, increase the variance; if it is less, decrease
the mutation variance."

The $(\mu + 1)$ algorithm was the first multimembered evolution strategy,
introduced by Rechenberg. The strategy introduced the concept of the pop-
ulation and the recombination operator. The method uses $\mu > 1$ parents
to produce one offspring. Two parents, $a = (x_a, \sigma_a)$ and $b = (x_b, \sigma_b)$, are
selected by the recombination operator. A new individual is then formed from these two by the random process:

\[
x_i' = \begin{cases} 
    x_a, & \text{if } \chi \leq 1/2 \\
    x_b, & \text{if } \chi > 1/2 
\end{cases} \\
\sigma_i' = \begin{cases} 
    \sigma_a, & \text{if } \chi \leq 1/2 \\
    \sigma_b, & \text{if } \chi > 1/2 
\end{cases} \tag{19}
\]

where \( \chi \) is an uniformly distributed random number on the interval \([0, 1]\). The object parameter \( x_i' \) of the new individual \( c = (x_i', \sigma_i') \) is then mutated in the same way as in the \((1 + 1)\) method, by equation (18). The deviations or the strategy parameters are not affected by the mutation but are set manually to comply with the 1/5 success rule. The selection operator then removes the least fit individual and the process is repeated.

Schwefel extended the \((\mu + 1)\) strategy to include \( \lambda > 1 \) children by the development of the \((\mu + \lambda)\) and the \((\mu, \lambda)\) methods [24]. The difference between these methods is the life span of each individual. The \((\mu + \lambda)\) strategy allows the solutions to survive from one generation to the next, which the \((\mu, \lambda)\) does not.

The main improvement of these two methods is the self-adaptation of the strategy parameters. This is done by including the deviation parameters in the mutation process, so that the new mutation operator becomes:

\[
\sigma_i'' = \sigma_i' \exp N_{0, \delta \sigma} \\
x_i'' = x_i' + N_{0, \sigma_i''}. \tag{20}
\]

In these schemes the deviations are considered to be a part of the genetic material.

The idea is that individuals with good strategy parameters are more likely to produce vigorous offspring than those without and therefore the strategy parameters will adapt to the current situation. In the beginning of a search, the strategy parameters will be relatively large but as the algorithm comes closer to an optimum they will gradually shrink.

### 2.4.1 Recombination

One way of explaining the beneficial effects with recombination is to study the increased number of possible starting points in the search space. Without recombination every new generation has access to \( \mu \) possible starting points. If we introduce recombination between two parents that number is increased to

\[
\mu^2 + \mu(\mu - 1) \sum_{i=1}^{n-2} 2^i,
\]
where $n$ is the number of object parameters. By including all parents in the recombination the number of possibilities is further increased, this time to $\mu^n$.

There are two main types of recombination operators, the discrete and the intermediate as described by equation (21).

$$x'_i = \begin{cases} x_{a,i} \text{ or } x_{b,i} & \text{discrete} \\ \frac{1}{2}(x_{a,i} + x_{b,i}) & \text{intermediate.} \end{cases} \quad (21)$$

The intermediate operator has a tendency to reduce the genetic diversity of a population. However when operated on the strategy parameters it sometimes has the beneficial effect of reverting unsuitable extinctions. When a strategy parameter mutates to near zero value, it often recombines with a non zero strategy parameter and the extinction is reverted. Schwefel found it beneficial using different recombination operators for the object parameters and the strategy parameters, especially when using a discrete operator for the object parameters and an intermediate for the strategy parameters [24].

### 2.4.2 Self-adaption

What is the probability of an offspring being better adapted to the environment than its parent?

Let the environment be simulated by the sphere model shown in figure 5 on the next page and let the offspring be generated by a mutation process according to equation (18) on page 16. Then if we let $\sigma$ tend to zero, the probability of a successful mutation $P_s$ will converge to $1/2$, but the rate of convergence to the optimum, $\varphi$, will tend toward zero. If we let $\sigma$ tend to infinity, $P_s$ will tend to zero and $\varphi$ will again converge toward zero.

$$\sigma \to 0 \quad \Rightarrow \quad P_s \to 1/2 \quad \Rightarrow \quad \varphi \to 0$$

$$\sigma \to \infty \quad \Rightarrow \quad P_s \to 0 \quad \Rightarrow \quad \varphi \to 0$$

So the optimal value of $\sigma$ must be somewhere between the extremities zero and infinity.

When Rechenberg calculated the 1/5 success rule he used two different object functions, the corridor model $f_1$ and the sphere model $f_2$.

$$f_1(x) = c_0 + c_1 x \quad \forall i \in \{2, \ldots, n\} : -b/2 \leq x_i \leq b/2$$

$$f_2(x) = \sum_{i=1}^n x_i^2$$

The corridor model represents a simple linear function with inequality constraints, improvement is only achieved by moving along the first axis inside
a corridor of width \( b \). The object function of the sphere model is one of the simplest non-linear, unimodal functions.

**(a)** The sphere object function, the value of the function is increasing toward the center of the plot. **(b)** The corridor object function, the value of the function is increasing toward the upper right corner and the constraints are marked with the two courser lines.

**Figure 5.** Contours of the sphere and corridor object functions.

The expectations of the rates of convergence, \( \varphi \), for these functions are [20]:

\[
\varphi_1 = \frac{\sigma}{\sqrt{2\pi}} \left( 1 - \sqrt{\frac{2\sigma}{\pi nb}} \right)^{n-1}, \quad \text{for } n \gg 1
\]

\[
\varphi_2 = \frac{\sigma}{\sqrt{2\pi}} \left( \exp \left( -\left( \frac{n\sigma}{\sqrt{8r}} \right)^2 \right) \right) - \frac{\sigma}{\sqrt{2\pi}} \left( \sqrt{\frac{n\sigma}{\sqrt{8r}}} \left( 1 - \text{erf} \left( \frac{n\sigma}{\sqrt{8r}} \right) \right) \right)
\]

for \( n \gg 1 \),

where \( r \) denotes the current euclidean distance from the optimum.

From (23) it is possible to determine the optimal standard deviations \( \sigma_i^{\text{opt}} \) according to \( \frac{d\varphi_i}{d\sigma_i} \bigg|_{\sigma_i^{\text{opt}}, \varphi_i^{\max}} = 0 \):

\[
\sigma_1^{\text{opt}} = \sqrt{\frac{2}{\pi}} \cdot \frac{b}{n}; \quad \varphi_1^{\max} = \frac{1}{2\pi} \cdot \frac{b}{n}
\]

\[
\sigma_2^{\text{opt}} \approx 1.224 \cdot \frac{r}{n}; \quad \varphi_2^{\max} \approx 0.2025 \cdot \frac{r}{n}
\]

The probabilities for successful mutations are:

\[
p_1 = \frac{1}{2} \left( 1 - \sqrt{\frac{\sigma}{\pi b}} \right)^{n-1} \quad \text{for } n \gg 1
\]

\[
p_2 = \frac{1}{2} \left( 1 - \text{erf} \left( \frac{n\sigma}{\sqrt{8r}} \right) \right) \quad \text{for } n \gg 1
\]

Combining these probabilities with the optimal standard deviations yields: \( p_1^{\text{opt}} = \frac{1}{2\pi} \approx 0.184 \) and \( p_2^{\text{opt}} \approx 0.270 \). From these findings Rechenberg postulated his 1/5 success rule, 1/5 being approximately the mean of \( p_1^{\text{opt}} \) and \( p_2^{\text{opt}} \).
When an evolutionary algorithm searches for an optimum, it zigzags through the parameter space. With only one strategy parameter the step size becomes the same in all directions. This results in the offspring being distributed on a circular area around the parent, or equivalent to a circular area for a higher ordered search space, if the effects of the recombination is disregarded.

In the case of individual strategy parameters the offspring will instead be distributed on an elliptic surface around the parent, as can bee seen in figure 6. This ellipse will have its axis aligned with the axis of the parameter space. However, if the gradients are of similar inclination in both directions, the resulting distribution will again be circular. This can be seen in (b) in figure 6, where the lower of the three points has a circular distribution.

There exists methods to improve those distributions, these methods calculate correlations between the strategy parameters and are able to align the axis of the elliptic distributions so that its major axis points in the same direction as the gradient, see (c) in figure 6.

![Figure 6.](image)

Figure 6. In the three figures above, the contours of a two dimensional landscape with multiple local maximums is shown. In each of the figures three ellipses, marked with courser lines, are also shown. These ellipses show the variance of the probability distribution that determines where the next generation will be located. The parents of the next generation is located in the center of the ellipses.

A simple method to control the strategy parameters is to follow Rechenberg's 1/5 success rule and on a regular basis check the rate of successful mutations and adjust the standard deviations according to the success rule.

Another method is to see the strategy parameters as a part of the genetic material and let them recombine and mutate according to equation (21) on page 18 and equation (20) on page 17. Experiments [12] have shown the order of mutation to be of importance. The strategy parameters should be
mutated before the object parameters, so that the new strategy parameters can influence the mutations of the object parameters.

The idea is that the individuals with strategy parameters corresponding to the 1/5 - success rule will have the best chance of producing vigorous offspring and therefore pass on those parameters to the next generations. This results in the algorithm automatically controlling the transition between global and local search. As it closes on the optimum, it will gradually reduce the step sizes controlled by the strategy parameters.

Of the two basic multimembered evolution strategies the (μ, λ) is the one most suitable for self-adaption. What may happen in the (μ + λ) strategy is that an individual with well adapted object parameters but unsuitable strategy parameters might live through many generations without producing any vigorous offspring. This will not happen with the (μ, λ) strategy, where every individual only lives for one generation. If an individual with a good fitness value does not produce any vigorous offspring, it probably has poor strategy parameters and its genome will disappear from the gene pool.

Large fluctuations of the strategy parameters will reduce performance since the parameters will have less chance of complying with the 1/5 - success rule. One method to reduce the fluctuations are to use the intermediate recombination operator, equation (21) on page 18, for the strategy parameters. This method was discussed in the previous section and Schwefel found it successful [24].

2.4.3 Noise

Evolutionary algorithms are considered to be robust against noise. They are after all inspired by a very noisy optimization process, the natural evolution. Noise might even be helpful in some cases, helping the algorithm escape local maximum. Some research has been done regarding ES and noise, but there seems to be an agreement that more work is needed.

The main effects of noise are a reduction of convergence rate and a deterioration of the final optimum quality. Two primary techniques exist to improve ES with noisy object functions, resampling and population up-sizing.

Resampling means that every individual is evaluated a number of times and then the fitness value is calculated by averaging those results. Population up-sizing is achieved simply by increasing the size of the population. Both of these techniques require a lot of additional computations, especially since the evaluation usually is the part of an ES algorithm that consumes the most resources.

The question of which of these techniques that give the best performance improvement relative to the computational cost does not have a conclusive
answer. Resampling is supported by some research [6, 14] while population up-sampling is supported by other [11, 1].

The $\lambda/\mu$ ratio also seems to be an important aspect, Beyer [5] showed that in a noisy environment the optimal ratio should be $\hat{\mu} \approx \lambda/2$. Hammel and Bäck [14] used a population of $\mu = 15$ and $\lambda = 100$ leading to $\hat{\mu} \approx \lambda/6.77$, a fact that might have led them to draw the wrong conclusion.
3 Methods

The objective of the agent in the current study is to find the optimal long-term harvest rate of a fish population. The purpose of this thesis is to find parameter values that makes the agent achieve that objective.

To obtain these parameter values an evolutionary strategy complemented by a brute force algorithm is implemented.

In the beginning of this section the object parameters of the Evolutionary Strategy are briefly presented. Then different issues regarding the implementation of the ES are discussed. At the end of the section the brute force method is described.

3.1 Object Parameters

The parameters to be investigated and optimized are the number of RBFs, the step-size RBF, the step-size RL and the temperature. The other parameters are set manually or handled by the agent automatically. The number of radial basis functions, and the step-size RBF parameter are part of artificial neuron network that builds the agents internal model of the environment. The remaining object parameters i.e. the step-size RL and the temperature controls the reinforcement learning algorithm.

The step-size RBF is part of the layer-weight matrix update function shown below:

\[
    w_{jk}(t + 1) = w_{jk}(t) + \delta \frac{\partial E}{\partial w_{jk}(t)}
\]

where \( \delta \) is the stepsize.

Equation (27) below shows the update function for Watkins Q-learning which is used by the agent:

\[
    Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, s_t)]
\]

where \( \alpha \) is the stepsize.

The temperature is the parameter that controls the balance between exploration and exploitation. The agent uses a softmax method with a Boltzmann probability distribution to decide which action to make. The distribution is shown below in equation (28) where the temperature is represented by \( \tau \).

\[
    \frac{\exp(Q_t(s, a) / \tau)}{\sum_b \exp(Q_t(s, b) / \tau)}
\]

A high temperature will generate a nearly equal probability for every action being selected, while a low temperature will make the softmax policy selec...
behave very greedy, only selecting those actions with the highest Q-value. In the case of a zero temperature the action selection will be completely greedy, selecting only the best action known at the moment.

3.2 An ES Algorithm

Evolutionary strategies compared to other evolutionary algorithms were initially developed for real valued parameter optimization. Therefore it was naturally to choose an ES algorithm. The multimembered algorithms seemed appealing because they utilize self-adaptation of the strategy parameters and have been subject to much research.

However, there exists variations of the multimembered ES algorithms ($\mu$, $\lambda$) and ($\mu + \lambda$). Both will work with or without recombination. The number of individuals, that participate in the recombination can vary and the recombination can be discrete or intermediate. The ratio of $\mu$ and $\lambda$ must be decided and in order to reduce the effects of noise resampling or population up-sizing might be used.

The pseudo code of the ($\mu \dagger \lambda$) algorithms that are shown in figure 7 on the following page will now be briefly described. The notation ($\mu \dagger \lambda$) is used to represent either a ($\mu$, $\lambda$) or a ($\mu + \lambda$) ES algorithm without specifying which one.

The population $P(t)$ consists of vectors of object parameters ($\vec{x}_t$), strategy parameters ($\vec{\sigma}_t$), and fitness values ($f_t$). When the population is initialized, uniformly distributed random numbers are assigned to the parameters. The parameters have a permitted interval and that interval is used by the distribution.

The RBFs for example are allowed to be on the interval [1 100], because the agent needs at least one RBF in order to function and not to risk excessive computational time, an upper limit of 100 is selected.

The population is then evaluated and each individual, or solution, is assigned a fitness value, based on its performance. Thereafter the parents of the next generation are selected based on their fitness. They undergo recombination and mutation to generate the next population.

Either the parents are eliminated from the new population as in the ($\mu$, $\lambda$) method or they are passed on as in the ($\mu + \lambda$) method. The population is then evaluated again and the process repeats until some termination criteria are met.
3 METHODS

3.2 An ES Algorithm

\begin{algorithm}
\begin{algorithmic}
\STATE \textbf{Begin}
\STATE \hspace{1em} $t := 0$;
\STATE \hspace{1em} \textbf{initialize} \hspace{1em} $P(t) = (\bar{x}_t^1, \bar{\sigma}_t^1, f_t^1)$;
\STATE \hspace{1em} \textbf{evaluate} \hspace{1em} $P(t) = (\bar{x}_t^1, \bar{\sigma}_t^1, F(\bar{x}_t^1))$
\FOR {\textbf{not terminate}}
\STATE \hspace{1em} \textbf{select} \hspace{1em} $P(t + 1) = (\bar{x}_{t+1}^1, \bar{\sigma}_{t+1}^1, f_{t+1}^1) = (\bar{x}_t^1, \bar{\sigma}_t^1, f_t^1)$
\STATE \hspace{1em} \textbf{recombine} \hspace{1em} $P'(t) = (\bar{x}_t^1, \bar{\sigma}_t^1, f_t^1)$
\STATE \hspace{1em} \textbf{mutate} \hspace{1em} $P''(t) = (\bar{x}_t^1, \bar{\sigma}_t^1, f_t^1)$
\STATE \hspace{1em} \textbf{evaluate} \hspace{1em} $P''(t) = (\bar{x}_t^1, \bar{\sigma}_t^1, F(\bar{x}_t^1))$
\ENDFOR
\STATE \textbf{End}
\end{algorithmic}
\end{algorithm}

\textbf{Figure 7.} Pseudo-code of $(\mu + \lambda) - \text{ES}$, where $P(t)$ represents the population, $x_i^t$ the object parameter, $\sigma_i^t$ the strategy parameter and $f_i^t$ the fitness value.

3.2.1 ES Parameters

Besides the different variations of recombination, which will be discussed later, the population size and the ratio of $\mu$ and $\lambda$ are the most important aspects when implementing the algorithm.

To minimize the effects of noise, two techniques have been discussed, population up-sizing and resampling. Both techniques are supported by different research, hence due to implementation issues population up-sizing was chosen and the size of the population was set to 100 individuals.

Research performed by Beyer [5] suggested that the ratio $\mu$ and $\lambda$ are of importance when an ES algorithm has a noisy object function. Beyer calculated an optimal ratio of approximately 2. A ratio of 2 deviates considerably from the recommended ratio in the noiseless case which is approximately 6.77 [23], therefore a ratio of 2 was chosen according to Beyer's findings.

No special termination criteria were used but the simulation was allowed to run for fifty generations before it was terminated.

3.2.2 Recombination

There are many choices of how to implement recombination in $(\mu + \lambda)$ algorithms. The recombination can be discrete or intermediate as shown in equation (29).

\[
x_i' = \begin{cases} 
  x_{a,i} \text{ or } x_{b,i} & \text{ discrete} \\
  \frac{1}{2}(x_{a,i} + x_{b,i}) & \text{ intermediate.} 
\end{cases}
\] (29)
The number of parents involved can vary, in the equation above two parents participate in the creation of a new offspring. Another possibility is to chose new parents for every object parameter. The new parents can be drawn from the entire group of parents or from a smaller subgroup.

The method of recombining strategy and object parameters does not even have to be the same. Often the best results are achieved when using different recombination types. Especially by using discrete recombination for the object parameters and intermediate for the strategy parameters.

An explanation of this is that the intermediate recombination reduces the genetic diversity of the object parameters, which is not a desired effect. But when it operates on the strategy parameters it reduces fluctuations and helps to maintain the values of the strategy parameters, so that the evolution process complies with Rechenbergs 1/5 - success rule.

Another possibility is to ignore the recombination operation all together and let the mutation be the only search method. But as discussed in section 2.4.1 on page 17 the recombination introduces a beneficial genetic diversity to the population. The possible starting points of the search are increased from $\mu$ possibilities with no recombination to $\mu^2 + \mu(\mu - 1) \sum_{i=1}^{n-2} 2^i$, with a two parent recombination scheme.

Four different strategies were tested. Two without recombination, $(\mu + \lambda)$, and two with, $(\mu/2d2i + \lambda)$. The $(\mu/2d2i + \lambda)$ strategies use discrete recombination of the object parameters and intermediate of the strategy parameters. In both cases the recombination involves two parents.

### 3.2.3 Fitness Functions

When an individual is evaluated, an agent is created with parameters taken from the object parameters of that individual. The agent then runs for one episode and obtains a fitness value based on its performance.

Two different fitness functions are going to be investigated, they are based on the following two ecology equations:

1. Logistic growth with constant effort harvesting

\[
F(S) = \frac{dx}{dt} = I + rx(1 - \frac{x}{K}) - Ex
\]  

2. Logistic growth with a Monod harvest function

\[
F(S) = \frac{dx}{dt} = I + r x(1 - \frac{x}{K}) - E \frac{x}{k + x}
\]
Where $I$ is the immigration constant, $r$ the intrinsic growth rate, $K$ the carrying capacity, $E$ the effort and $k$ the half saturation constant. An immigration constant is added to the ecology functions, to prevent a collapse of populations being subjected to excessively hard harvest pressures. The immigration constant is relatively small and has only significant influence when the population is close to extinction.

The effort parameter $(E)$ is the only parameter controlled by the agent. And the objective of the agent is to find an optimal value for this parameter. In the first scenario (constant yield harvest) the agent has to learn that a low immediate return might lead to larger returns in the future. There exist no risk of the population collapsing if the harvest pressure is slightly too high.

In the second scenario (Monod harvest function) however, the risk of collapse exist. The agent still has to learn how to manage the size of the population so that it is close to $x_{max}$, where the population has its maximum growth rate. However, if the harvest pressure becomes slightly too high and the agent does not rapidly decrease the pressure the population will collapse.

The noise in fitness functions does not originate from the ecology functions, which are deterministic. However when the agent is initiated, it randomly distributes the positions of the radial basis functions and its action selection uses a probability distribution. These are the factors that introduce noise into the fitness function.

One episode of the fitness functions is 300 time-steps long. During that time the intrinsic growth rate is changed twice, to force the agent to relearn its environment. The intrinsic growth rate starts at a value of 1, at time-step 100 it is lowered to $3/2$ and at time-step 200 it is raised to $1/2$.

### 3.3 A Brute Force Algorithm

To investigate the results from the evolution strategies a simple brute force method was developed. The parameter values attained from the ES algorithms served as the starting point for the algorithm. The parameters were then changed one at a time, while the other remained at their original value. Every parameter was evaluated on 200 equidistantly distributed points on the interval $[0, 100]$ and every point was averaged over five evaluations.
4 Results

Results from four different evolution strategies simulated with the ecology modeled by logistic growth and a constant effort harvest function are shown in figure 8.

![Graphs showing results](image)

**Figure 8.** Results obtained using a constant effort harvest function. Each sub-figure is split into five separate plots. The top plot shows the development of the fitness of the best individual (solid line) and the average fitness of the population (dotted line). The other four sub-plots show the development of the average value of the parameters (solid line) and their variances (dotted line).

The maximum and average fitness values level off at high values for all strategies except for the \((\mu/2d2i + \lambda)\) strategy. For the strategies without recombination all of the parameters stabilize and their variances decline to
almost zero.

The parameter values of the \((\mu/2d2i, \lambda)\) strategy stabilized. However, the variances of the parameters did not decline significantly except for the step-size RL and to some degree the Step-size RBF. The \((\mu/2d2i, \lambda)\) strategy did not perform well, the parameter values did not stabilize sufficiently and the variances of the parameters did not decline considerably.

\[ Figs. \text{ (a)-(d)} \] \text{ Results obtained using a Monod harvest function.} Each sub-figure is split into five separate plots. The top plot shows the development of the fitness of the best individual (solid line) and the average fitness of the population (dotted line). The other four sub-plots show the development of the average value of the parameters (solid line) and their variances (dotted line).

Figure 9 shows the result from four different simulations with an ecology modeled by logistic growth and a Monod harvest function. The figure shows
that none of the maximum fitness values has stabilized. The maximum fitness oscillated; for some generations it was high but in later generations it deteriorated. The average fitness stabilized but it did so at low fitness values. The step-size RBF and RL parameters stabilized for all strategies but none of the other parameters did.

Figure 10 on the next page shows plots with information of two agents operating on an ecology modeled by the Monod harvest function. The best agent of generation 49 shown in sub-figure (a) was quite successful while the best agent of generation 50 was not.

The successful agent managed to avoid a collapse of the population when the intrinsic growth rate changed at time step 100. The size of the population and the rewards decline after the change of the growth rate but the agent manages to reduce its harvest rate before the population collapses.

When the second change of the intrinsic growth rate takes place the agent relearns the new environment rapidly, the size of the population only shortly surpasses the optimal size and is then stabilized by the agent. The agents internal model of the environment shown by the surface plot in figure 10 on the following page has a distinctive peak at the correct location for the last value of the intrinsic growth rate.

The unsuccessful agent did not manage to avoid a collapse when the growth rate was changed the first time. At the second change the population recovers but the agent never finds the optimal harvest rate. This is also seen in the plot of the agents internal model, where two distinct peaks are seen but none of them is at the right position.
Figure 10. The best individual of generation 49 and generation 50 of the $(\mu + \lambda)$ strategy with a Monod harvest function (sub-figure (b) in figure 9 on page 29). In the top graph the status of the population and the received rewards are displayed. The graph beneath shows which action the agent selected at each time step. At time step 100 and 200 the intrinsic growth rate is changed, which forces the agent to relearn the maximum sustainable yield. The surface plot to the right shows the internal model of the agent at the last time step.
The results from the brute force algorithm is shown in figure 11. Optimal values of all parameters except the temperature are indicated by the figure. With exception of the temperature, the parameter values suggested by the brute force algorithm concur with those indicated by the evolution strategys in figure 8 on page 28.

![Graphs showing fitness vs parameter value for RBFs, Step-size RBF, Step-size RL, and Temperature.](image)

Figure 11. Results from the brute force algorithm. The default parameter values are set to: RBFs = 65, Step-size RBF = 10, Step-size RL = 1 and the Temperature = 60

Figure 12 on the next page shows parameters plotted against the fitness of the individual that the parameter belonged to. It can be seen that strains of successful individuals have formed, each strain with different parameter values. Except for the step-size RL parameter where all successful individuals belonged to the same strain.

Two simulations with the Monod harvest function and the interval of the temperature reduced to [0 20] were performed, the results are shown in figure 12 on the following page. The fitness values leveled off at high values and the parameters stabilized. The variances of all parameters except the temperature declined to almost zero.
Figure 12. The fitness plotted against the different object parameters. Each individual of the simulation is plotted in the graphs. The figure shows the result from the $(\mu + \lambda)$ strategy with the Monod harvest function.
Figure 13. Results obtained using a Monod harvest function. Note that the temperature interval is reduced to $[0, 20]$. Each sub-figure is split into five separate plots. The top plot shows the development of the fitness of the best individual (solid line) and the average fitness of the population (dotted line). The other four sub-plots show the development of the average value of the parameters (solid line) and their variances (dotted line).
5 Discussion

Of the two different ecological models the one using the Monod harvest function presented the largest challenge. In contrast to the constant effort harvest it is sensitive to overfishing. The agent does not only has to learn the maximum sustainable yield, it also has to learn how to avoid a population collapse.

What we would like to see in the simulations is the average fitness and the fitness of the best individual to level out at high values. The parameter values should stabilize and their variance should decline to almost zero. If that is achieved, we can say that the strategy has found a good set of parameter values.

5.1 Constant Effort Harvest

Out of the four evolution strategies tried on the ecology with constant effort harvest, three performed well and the fourth relatively well. Surprisingly the two strategies without recombination displayed the best performance, and they delivered conclusive results. Of the two strategies without recombination the \((\mu + \lambda)\) had the best performance.

The results of the \((\mu + \lambda)\) simulation shown in figure 8 on page 28 show that all parameters have stabilized and their variances have declined to almost zero at the end of the simulation.

The temperature is relatively high, this might be important because of the changing environment during an episode. The intrinsic growth rate of the logistic growth is changed twice during an episode and a high temperature makes the agent more explorative, which is beneficial when the environment has to be relearned.

5.2 Monod Harvest Function

None of the original strategies performed well on the ecology with a Monod harvest function. However, the strategies without recombination still performed better then those with.

The \((\mu + \lambda)\) strategy was the one that performed best out of the four tested. Initially it performed well, the fitness of the best individual rose and leveled off at a relatively high value. However, at generation 34 the fitness dropped and started to oscillate. The average fitness of the population also rose initially but it leveled off at a relatively low value.

The brute force algorithm shown in figure 11 on page 32, indicates good values for the RBF, the Step-size RBF and the Step-size RL parameters.
However, no indication of which values might be good for the temperature parameter could be obtained by the algorithm.

If we look at the temperature in figure 12 on page 33 there seem to be two strains of solutions, one strain with a temperature around 18 and another with a temperature around 70. The majority of the population appears to have belonged to the 70 strain.

And if we look at the best individuals of the two last generations, shown in figure 10 on page 31, we can see that they are off different strains, the best individual of generation 49 belongs to the 20 strain and the best individual of generation 50 belongs to the 70 strain. It is the individual of the 20 strain that obviously excels of the two. This indicates that a lower temperature is probably beneficial.

The comparison of only two individuals certainly is not enough to determine the value of the temperature parameter. However, it is enough to investigate the performance of the agent if we force the temperature to a lower value.

Figure 13 on page 34 shows the results from two simulations when the upper limit of the temperature interval is lowered to 20. It is still the $(\mu + \lambda)$ strategy that performs best and now it performs very well.

At the end of the simulation shown in figure 13 on page 34, all parameters except the temperature have stabilized, the variance of the temperature is almost as high as in the beginning of the simulation. However, the temperature is still much lower than it was in the first simulations when the interval was larger.

The best simulations of the two different ecologies are shown in figure 14 on the next page for comparison of the parameter values. The number of RBFs and the step sizes of the RBF network and RL algorithm are similar, however the temperature value is very different.

Because of the sensitivity too excessive harvest rates in the ecology with a Monod harvest function the temperature has to be low. If the agent has found the maximum sustainable yield and dramatically increases the harvest rate due to a high temperature the population might collapse. If the temperature is low, the harvest rate is not changed as rapidly and the agent has longer time to respond and decrease the harvest pressure to avoid the collapse.
(a) $(\mu + \lambda)$ - Constant Effort Harvest function. (b) $(\mu + \lambda)$ - Monod harvest Function.

Figure 14. The most successful simulations of the two different ecologies.
6 Conclusions

The evolutionary optimization algorithm successfully found parameter values that were well adapted for the agent when it managed resources modeled by both of the ecology functions. For the resources modeled by logistic growth and a constant effort harvest function it was unproblematic to find good parameter values. However, the resources modeled by the Monod harvest function presented the evolutionary algorithms with a tougher challenge.

Initially none of the strategies performed well, after analysis of the parameter values two strains of solutions were found, one with high temperature values and one with low. The high value strain performed worse than the low value strain, but nonetheless it constituted a majority of the population. After limiting the permitted interval of the temperature parameter to exclude the high value strain, the algorithm found parameter values that were well adapted.

After the interval was limited, the \((\mu + \lambda)\) strategy without recombination showed the best results of both of the ecologies.

This thesis only investigated four out of many parameters that control the agents behavior. However, the tool developed is adaptable and might easily be used to optimize and analyze other parameters.

The fitness function might also be modified, allowing the agent to be optimized for other behaviors than the optimal management, which has been the aim of this thesis.

To further improve the optimization tool resampling and correlated mutations might be added to the algorithm, perhaps those techniques might solve the problem that was encountered with the temperature parameter.
References


REFERENCES


