Interconnected Learning Automata Playing Iterated Prisoner's Dilemma

by

Henning Hetland
Tor-Øyvind Lohne Eriksen

Masters Thesis in
Information and Communication Technology

Agder University College

Grimstad, June 2004

Summary

There exist several examples of situations where interaction between entities is essential in order to reach a common goal, for instance negotiations. Problems arise when the parties of a case are in disagreement with each other and they all want the best for themselves. Our thesis focuses around one such case. Analyzing the results of its own actions, an entity can learn what kind of adversaries it faces. For instance they may be cooperative, hostile, ignorant or irrational. The situation becomes even more complex when the opposing parties change its behaviour while trying to reach a goal. In this case one has to detect and adapt the change of behaviour.

A "Learning Automaton" (LA) is a learning entity that learns the optimal action to use from its set of possible actions. It does this by performing actions toward an environment and analyzes the resulting response.

In our thesis the environment is the game of "Iterated Prisoner's Dilemma" (IPD) which is a non-cooperative, non-zerosum game. This game represents a case where two prisoners are held in prison of a common theft. They are given the option to either defect and turn against the other prisoner by telling the truth or to cooperate with the other prisoner by holding the truth back. The game is played over several iterations where actions performed in the past may affect the present choice of actions.

There is no set optimal strategy in this game. The optimal action depends on the strategy of the opponent. If the opponent should switch strategy during play, we get a nonstationary environment. This makes the situation even more challenging, since the learner will have to adapt toward new strategies when the opponent switches. State-of-the art technologies do not handle such nonstationary environment well. However, nonstationary environments are in accordance with most real-life situations; environments are seldom static. The main aim of our thesis is to design systems of Learning Automata that handle IPD games where the opponent switches strategy during the game.

It is our hypothesis that "Interconnected Learning Automata" (ILA) will play better in such an environment than other existing automata. Interconnected learning automata are several learning entities connected in a network to achieve a common task. If the learner also remembers previous moves, we believe the learner will be able to adapt to rapid changes in the environment quickly.

To evaluate this we have developed three new players. One player remembers the moves performed in recent iterations in order analyze the opponent more effectively. This player is called SLASH. The second player is called the Hierarchical player where a learner has a selection of predefined strategies as its action set. The third player is called I-SLASH which consists of several SLASH players arranged in a hierarchy like the Hierarchical player.

A prototype was developed to evaluate the new players in a nonstationary environment. Several tests where defined to evaluate how well the new players perform compared to other

well known players. There have also been tests evaluating how well the new players play comparing to the optimal play.

Tests show that the Hierarchical player does well, but will always be limited to its set of available strategies. The Hierarchical player also has a slow learning rate because every strategy will have to be tested in turn. SLASH shows a remarkable ability to learn to play close to optimal in most cases. In the tests performed SLASH does well overall, but is beaten by its subspecies I-SLASH. I-SLASH has a higher learning rate than SLASH and receives on average a better score in most of the cases we have tested.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree Sivilingeniør / Master of Science at Agder University College, Faculty of Engineering and Science. This work was carried out under the supervision of cand.scient. Ole-Christoffer Granmo. The work has been done in the period January – May. We have done all the work at Agder University College.

First of all, we would like to thank our supervisor, cand.scient. Ole-Christoffer Granmo who introduced us the area of learning automata. We have learned a great deal from Granmo, and he has been supportive all the way. We would also like to thank all our wonderful co-students and the Head of Studies, Stein Bergsmark, for all his contributions.

Grimstad, May 2004

Tor-Øyvind Eriksen and Henning Hetland

Table of contents

List of figures

List of tables

1 Introduction

The idea of having computers doing services for humans has existed for several years. Over the last few decades this idea has become more and more a reality. As processor power and memory increases, machines are able to do more complex tasks. Not only are they capable of replacing human labour in areas considered health hazardous, but they can also help solving complex psychological tasks. This thesis is based around one such task, oriented around a game which can be mapped to real life situations.

The field of artificial intelligence is vast, concerning many fields. The main principle is to understand the nature of intelligent actions and then to construct computer systems capable of such actions [10]. Also according to [10] many activities involve intelligent action; problem solving, perception, learning, symbolic activity, creativity, language and so forth. These activities are shared by many fields, including those of psychology, linguistics and philosophy of mind, in addition to that of artificial intelligence.

Thathachar and Sastry [11] distinguish between two types of artificial intelligence. The first type is conventional AI. In this case a formal symbol manipulation process realizes the intelligence. Ideas are represented with symbols, meaning that intelligence is given from computing with these symbols using some formal rules. The second type is neural networks. The idea is to imitate the human brain, where the brain is looked upon as a network of neurons. By making artificial networks of artificial neurons one wants to achieve the learning capacity of the brain.

In either case learning is essential. In many cases, learning can be achieved through analyzing responses from an environment. Learning automaton is one of many solutions that can be applied to achieve this. A learning automaton learns from the response given to it by the environment. The response, being both good and bad, results in behaviour change to the automaton. Thathachar and Sastry [11] mentions three different ways the environment can add to the learning of an external entity. The environment may have exact knowledge of what action is to be considered correct and will guide the external entity toward this action. This type of learning is called supervised learning. Another solution has an environment that will not tell the learner what is the best action. It will however evaluate the actions made by the learner and give a scalable feedback to the learner. The scalar feedback tells the learner how good the chosen action was. It is then up to the learner to analyze all actions available and learn from the scalar feedback. This type of learning is called reinforcement learning. If this feedback is subject to change over time for the same actions, making it a random variable, the environment itself becomes random. Random environment is the third way the environment can add to the learning of a learner. In this case the learner will have to try all actions a number of times in order to determine the mean reinforcement associated with each action and then find the best action.

The Prisoner's Dilemma is a game with simple rules, but which can be mapped toward an interestingly high amount of real life situations, for instance nuclear disarmament [9]. What makes a simple game like this so complex is that there is no real optimal solution. This

comes especially true if the game is run over several iterations. It all depends on how both players work together or put another way; what strategy they choose to follow.

The game of Prisoner's Dilemma gives an excellent situation to analyze with a learning entity. Since there is no optimal solution (although it can be argued that the rational choice is to defect) one can not analyze this case with supervised learning. The environment simply can not tell what actions are optimal. The environment can however give a feedback of what is good and bad based on the rules of the game. This makes for a situation where reinforcement learning can be used for analyzing the game. The environment can now check the actions of both players and give a feedback of these actions based on some set of predefined rules. The learners will use this feedback to reinforce its set of probabilities of choosing an action.

Analyzing the Prisoner's Dilemma using learning entities is not a new idea. For instance Kehagias [7] uses a probabilistic learner, called a learning automaton, to analyze the Prisoner's Dilemma over several iterations.

In this thesis the problem is taken further. What if the opponent player should change strategy during play? This opens for a nonstationary environment and adds more challenges to the learner. Since there is no optimal play for the game overall, there is usually an optimal play against an opponent strategy. If the opponent strategy switches, then the learner will have to notice this switch and choose new actions.

The minority of real life situations appears in a static environment, so efficient learning in a dynamic switching environment is vital. We will in this thesis present possible improvements to learning techniques. We will also describe how we have used these techniques to develop new learning automata, and how well they perform in a nonstationary environment.

Several solutions exist to make an automaton more efficient in such environment. This thesis focuses on two techniques that may achieve additional efficiency. The first technique is to remember the previous moves an opponent makes. The idea is to achieve better learning by having the learner recognise patterns in the moves made in the past, both by the player and the opponent. The second technique is to organize the player in a hierarchy where the player can choose from several strategies. It can then choose the most optimal strategy against any opponent at any time.

Our thesis description deals with these topics and is summarized in the next section.

1.1 *Thesis definition*

The overall goal for a learning automaton playing the Iterated Prisoner's Dilemma will be to play as optimal as possible given the opponent(s). This requires in many cases a mutual cooperation between the players, and in other cases to exploit the goodness of the opponent player. If the environment is nonstationary, the opponent will change its strategy during the game. An optimal player against one particular strategy may not be able to play well against another strategy. The goal of this project will be to examine whether interconnected learning automata will be more suitable to play the Iterated Prisoner's Dilemma in a nonstationary environment compared to existing automata.

The final thesis definition is formulated like this:

The students will develop a prototype of learning automata which are interconnected. They will evaluate its behaviour in a nonstationary environment playing the Iterated Prisoner's Dilemma. They will also try to connect the interconnected automata with each other in a hierarchy. The students will look at the effects of using game history in learning automata.

1.2 *Report outline*

The report is structured with 5 main sections. The first section is the introduction which is the current chapter. The second section is theory that sets the fundament of the work we are going to do. Chapter 2 and 3 covers the theory. Chapter 2 introduces basic theory like the learning automaton and the game of Iterated Prisoner's Dilemma. Chapter 3 introduces more advanced topics like nonstationary environment and interconnected automata. The third section covers the method of how we work and what we have done. The goal of this section is to map the theory section over to something we have produced and can perform tests with. Chapter 4, 5 and 6 covers this section. Chapter 4 introduces in detail what we want to achieve with this thesis and how we want to achieve the goals. Chapter 5 introduces the solutions with which we are to perform tests with. Chapter 6 introduces the prototype developed in which the solutions will run. The fourth section covers the tests performed to evaluate the solutions up against the goals introduced in the method. This section is covered by chapter 7 and 8. Chapter 7 covers the results of the tests while chapter 8 discusses in detail the results. The last section is the conclusion which has its own chapter, namely chapter 9.

2 Learning Automata and the Iterated Prisoner's Dilemma

This is the first chapter of background theory. This chapter deals with the basic subjects. Learning automata is the first subject. This theory sets the foundation of the new players we are to define. The second subject is the Iterated Prisoner's Dilemma. The theory in this section introduces the environment in which the learning automata are to operate in.

2.1 *Learning Automata*

The study of learning automata began with the article [1] published by Tsetlin(1973). In his work he introduces a deterministic automaton operating in random environments as a model of learning [2]. The crucial keyword is learning. We will define learning as the ability to improve ones ability based on experience from previous behaviour. Learning automaton is one of many solutions that can be applied to achieve learning.

The second keyword is environment. A learning automaton learns from the response given to it by the environment. The response, being both good and bad, results in behaviour change to the automaton. This behaviour change will be discussed later and is often called reinforcement algorithm.

2.1.1 Environment

The environment provides the automatons ability to learn. By doing a specific action, the environment will reply with a favourable or unfavourable response. The automaton will learn based on this response. The environment must be unknown to the automaton; the automaton should just give an action to the environment and the environment will produce an outcome. External or internal changes to the environment are to be regarded as hidden.

In the literature the environment is often defined by a triple $\{\alpha, c, \beta\}$ where $\alpha = \{\alpha_1, \alpha_2,\ldots, \alpha_r\}$ represents a finite set of actions being the input to the environment, $\beta = \{\beta_1, \beta_2\}$ represents a binary output set, and $c = \{c_1, c_2,\ldots, c_r\}$ is a set of penalty probabilities, where c_i is the probability that action α_i will result in an unfavourable response. Given that $\beta(n) = 0$ is a favourable outcome and $\beta(n) = 1$ is an unfavourable outcome at time instant n (n = 0, 1, 2,…), the element c_i of c is defined mathematically by

$$Pr(\beta(n) = 1 \mid \alpha(n) = \alpha_i) = c_i \quad (i = 1, 2, \ldots, r)$$

There is no standard for what values of the output set are to represent unfavourable or favourable response. Different literature may set the value 1 to represent either favourable or unfavourable response. However, using 1 to represent unfavourable response seems to be most common and is also the standard used in [3].

The response values can be represented in three different models. In the P-model, the response values are either 0 or 1, in the S-model the response values is continuous in the range (0, 1) and in the Q-model the values is in a finite set of discrete values in the range (0,

1). The Q and S models, in other words, introduces varying penalties, meaning reward and punishment can be given to the automaton with different strength.

The environment can further be split up in two types, stationary and nonstationary. In a stationary environment the penalty probabilities will never change. In a nonstationary environment the penalties will change over time. This last case will be introduced in more detail later.

Figure 2.1.1 show a general environment

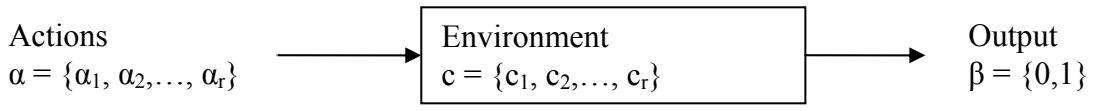

Figure 2.1.1

The figure shows a how the action set forms the input of the environment which result in an output depending on the penalty probabilities.

2.1.2 The Automaton

The automaton is the learning entity. In the simplest case an automaton only switches between two action states based on the outcome of the environment. Before looking at how an automaton works, we must have a good definition of describing an automaton. In the literature the automaton is defined by a quintuple $\{\Phi, \alpha, \beta, F(\bullet,\bullet), H(\bullet,\bullet)\}$

$\Phi = \{\Phi_1, \Phi_2,\ldots, \Phi_s\}$ is the set of internal states of the automaton. The internal states determine the action to be produced by the automaton. The state at time instant n is denoted by $\Phi(n)$ and is an element of the finite set Φ.

$\alpha = \{\alpha_1, \alpha_2,\ldots, \alpha_r\}$ denotes the set of actions that can be produced by the automaton. This is the output set of the automaton, hence also being the input set to the environment. The action done at time instant n is denoted $\alpha(n)$ and is an element of the finite set α.

$\beta = \{\beta_1, \beta_2,\ldots, \beta_m\}$ or $\beta = \{(a, b)\}$ is the input set to the automaton, that is the set of response from the environment. This set can be either finite or infinite. $\beta(n)$ denotes the input to the automaton at time instant n.

$F(\bullet,\bullet)$ is a function that maps the current state and the response from the environment into the next state, given mathematically by $F(\bullet,\bullet):\Phi \times \beta \rightarrow \Phi$. This formula is called the transition function. A similar expression is showed by the formula

$$\Phi(n+1) = F[\Phi(n), \beta(n)]$$

The transition function can be either deterministic or stochastic. If the function is deterministic the result of the function is uniquely specified for each state. An example is having the state Φ_1 and the automaton is given the response β_1 from the environment which

will always result in the next state being Φ_2. If the function is stochastic the next state could be one or several of the states in the set Φ, the result are dependent on a probability that each state with a probability higher than 0 is possible.

H(•,•) is the output function and is defined mathematically by $H(\bullet,\bullet):\Phi \times \beta \rightarrow \alpha$. This function maps the current state and the response from the environment into the action produced by the automaton. There are cases where the current input does not have any influence on the action being produced. This is called a state-output automaton; the action is decided from the current state only. In this case the function is $G(\bullet):\Phi \rightarrow \alpha$. A similar expression is showed by the formula
$\alpha(n) = G[\Phi(n)]$

This function can also be either deterministic or stochastic.
In short the automaton takes an input from the environment and produces an action based on this. The automaton is showed in Figure 2.1.2

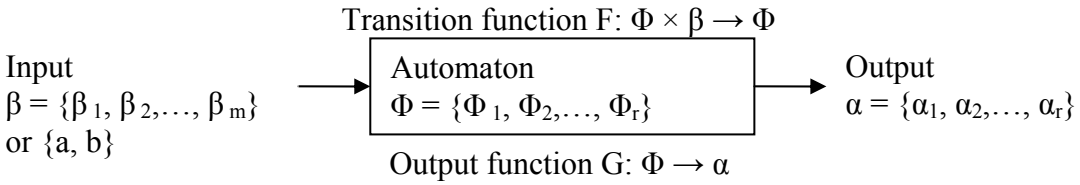

Figure 2.1.2

2.1.3 Stochastic automaton

An automaton is stochastic if either one of the functions F or G is stochastic. If F is stochastic, the elements of this set are denoted by f_{ij}^{β}. The value of this element represents the probability that the automaton moves from state Φ_i to Φ_j given the input β.

$$f_{ij}^{\beta} = Pr\{\Phi(n+1) = \Phi_j \mid \Phi(n) = \Phi_i, \beta(n) = \beta\} \qquad i,j = 1, 2, \ldots, s \ \beta = \beta_1, \beta_2, \ldots, \beta_m$$

If G is stochastic, the elements of this set are denoted g_{ij}. The value of this element represents the probability that the action done by the automaton is α_j given the automaton is in state Φ_i.

$$g_{ij} = Pr\{\alpha(n) = \alpha_i \mid \Phi(n) = \Phi_i\} \qquad i,j = 1, 2, \ldots, r$$

2.1.4 Fixed structure automata

The work [1] introduced by Tsetlin covered the topic of fixed structure automata. An automaton is called to be fixed structured when the functions f_{ij} and g_{ij} are having values that do not change over time. An example of this type of automata is the Two-state automaton $L_{2,2}$. This automaton consists of two states and can perform two different actions. As input it accepts 0 and 1 from the environment, where $\beta = 1$ is unfavourable and $\beta = 0$. If the response

from the environment is favourable the automaton does not change the current state, however in unfavourable situations the state is changed. These cases are illustrated in the Figure 2.1.3.

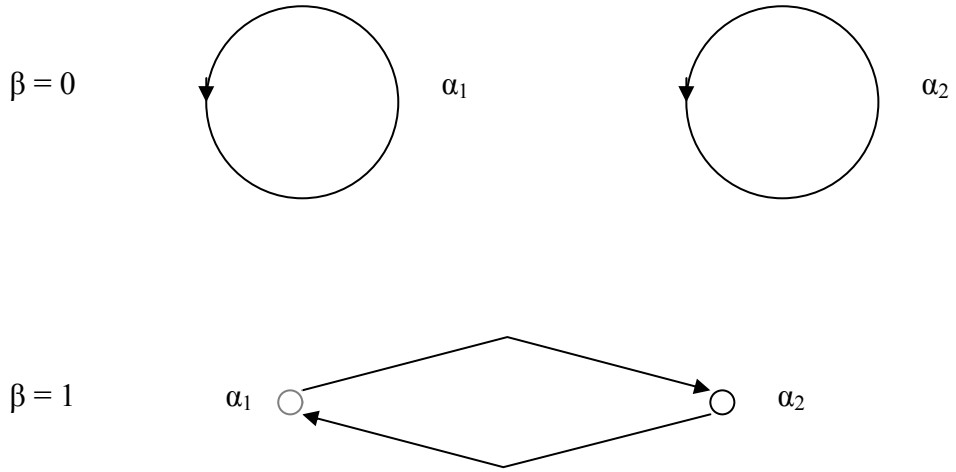

Figure 2.1.3

The environment consists of the penalty probabilities $\{c_1, c_2\}$ where c_i is the probability that action α_i results in an unfavourable outcome. These probabilities do never change. If the probability of choosing both α_1 and α_2 is the same, we have what is called a pure chance automaton.

2.1.5 Variable structure automata

In the latter case the probabilities were fixed. By making them change over time, one can get a greater flexibility where actions rewarded will get a higher chance of being chosen again. The state transitions or the action probabilities will be updated for each round using a reinforcement scheme. Several types of reinforcement schemes exist and discussing them all is beyond the scope of this thesis. However, this chapter will introduce some of the reinforcement schemes used throughout the thesis.

Both transition and action probabilities can be updated. [3] tell that "Varshavski and Vorontsova (1963) were the first to suggest automata that update transition probabilities. Later developments by Fu and his associates (1965, 1966, 1970) made extensive use of action probabilities as these were found to be mathematically more tractable."

A variable structured stochastic automata is represented by a quintuple $\{\Phi, \alpha, \beta, A, G\}$ where A represent the reinforcement scheme. The rest of the symbols are the same as above. Updating action probabilities can be represented mathematically by

$p(n+1) = T[p(n), \alpha(n), \beta(n)]$

where T is a mapping. This formula says the next action probability (p(n+1)) is updated based on the current probability (p(n)), the input from the environment and the resulting action.
Updating transition probabilities can be represented mathematically by

$f_{ij}^{\beta}(n+1) = T'[f_{ij}^{\beta}(n), \Phi(n), \Phi(n+1), \beta(n)]$

where T' is a mapping. The transition $f_{ij}^{\beta}(n+1)$ from state Φi to Φj with the input β is updated based on the transition $f_{ij}^{\beta}(n)$, the states of the automaton at time instances n and n+1 and the input β(n).

2.1.5.1 Reinforcement schemes

This chapter will only look at reinforcement schemes where action probabilities are updated. The reinforcement schemes are the basis of the learning in the variable structured automata. By being rewarded or penalized from the environment based on an action of the automaton, the probabilities of doing actions of the set α are updated. How this is done depends on the reinforcement schemes being used. Linear Reward Peanlty, Linear Reward Inaction, Linear Inaction Peanlty and the pursuit algorithms are schemes that will be covered further.

2.1.5.1.1 Linear Reward Penalty L_{R-P}

In this scheme the action probabilities are updated at every step. When a reward is given from the environment the probability of the current action is increased while the other probabilities are decreased. When a penalty is given from the environment the probability of the current action is decreased while the other probabilities are increased. The equations are as follows

$$\begin{aligned} &\alpha(n) = \alpha_i \\ &p_i(n+1) = p_i(n) + a[1-p_i(n)] && \beta(n) = 0 \\ &p_j(n+1) = (1-a)p_j(n), j \neq i \\ &p_i(n+1) = (1-b)p_i(n) && \beta(n) = 1 \\ &p_j(n+1) = (b/(r-1)) + (1-b)p_j(n), j \neq i \end{aligned}$$

a and b are learning rates that decides the intensity of the learning. a and b should have the same value, but it is possible to use different values to favour reward over penalty or vice versa. The ideal use of this scheme is getting an action to be chosen of a probability of 1 and this action being the most ideal in the long run.

2.1.5.1.2 Linear Reward Inaction L_{R-I}

This scheme is somewhat like the Linear Reward Penalty scheme, but it does not update the action probabilities upon penalties from the environment. The equations are as follows

$\alpha(n) = \alpha_i$
$p_i(n+1) = p_i(n) + a[1-p_i(n)]$ $\beta(n) = 0$
$p_j(n+1) = (1-a)p_j(n), j \neq i$
$p_i(n+1) = p_i(n)$ *for all i if* $\beta(n) = 1$

2.1.5.1.3 Linear Inaction Penalty L_{I-P}

This scheme is the opposite of the Linear Reward Inaction. The action probabilities are updated only when a penalty is given from the environment, rewards are ignored.

$\alpha(n) = \alpha_i$
$pi(n+1) = pi(n)$ *for all i if* $\beta(n) = 0$
$p_i(n+1) = (1-b)p_i(n)$ $\beta(n) = 1$
$p_j(n+1) = (b/(r-1)) + (1-b)p_j(n), j \neq i$

2.1.5.1.4 Pursuit algorithm

This algorithm is similar to the Linear Reward Inaction. The action probabilities are not updated when the response from the environment is unfavourable. However, if the response from the environment is favourable, the action with the highest estimate of reward is updated. This means that the action taken not necessarily will be given a higher probability of being repeated.

The equation is almost the same as for Linear Reward Inaction

$\alpha(n) = \alpha_i$
$p_k(n+1) = p_i(n) + a[1-p_i(n)]$ $\beta(n) = 0$
$p_j(n+1) = (1-a)p_j(n), j \neq k$
$p_i(n+1) = p_i(n)$ *for all i if* $\beta(n) = 1$

p_k is the action probability for the action of with the highest estimate of reward. Finding the estimate is done by keeping track of the number of time an action has been rewarded and how many times an action has been selected. The equation below describes this

$W_i(n+1) = W_i(n) + (1 - \beta(n))$
$Z_i(n+1) = Z_i(n) + 1$
$d_i'(n+1) = W_i(n+1) / Z_i(n+1)$

d_i' calculates which action has the highest probability of being rewarded, and is thus the estimator. It is calculated by keeping count of how many times the ith action has been rewarded up to n ($W_i(n)$) and how many times the ith action has been selected up to n ($Z_i(n)$).

2.2 *Prisoner's Dilemma*

Prisoner's Dilemma (PD) is a non-cooperative, non-zerosum game played between two players. A non-zerosum game denotes that one player's benefit not necessarily come at the expense of someone else, while in a zerosum game one player must lose before the other player can win. A non-cooperative game denotes that the players are unable to communicate prior to the game, and that each player is unable to know what the other player does before it decides its own action.

The classical Prisoner's Dilemma can be described as two persons, player A and player B, which are arrested by the police. They both know that if one confess and the other remains silent, the former will go free, while the other will get full time (5 years) in prison. If both stay silent they will get 2 years, while both confessing will lead to 4 years in prison. Each player has two choices, but regardless of what the other player does, the best output will be to confess. If both players think this way, they will end up with 4 years in prison, instead of 2 years, hence the dilemma.

This illustrates that the players would be better off if they where able to cooperate and coordinate their actions. In this thesis the terms *(C) Cooperate* and *(D) Defection* are used to illustrate the actions of each player, and the letters *R, S, T* and *P* are used to illustrate the different penalties. *(R) Reward* denotes the penalty for mutual cooperation, *(S) Sucker's* payoff denotes cooperation against a defective opponent, *(T) Temptation* denotes defection against a cooperative opponent, and (*P) Penalty* denotes mutual defection. The penalties may vary, but are constrained to two rules:

$2R < S + T$
$S > P > R > T$

Table 2.2.1 illustrates the game with cost names, while Table 2.2.2 illustrates the game with the penalty values used throughout this thesis.

		Player B	
		Cooperate	**Defect**
Player A	**Cooperate**	R, R	S, T
	Defect	T, S	P, P

Table 2.2.1

		Player B	
		Cooperate	**Defect**
Player A	**Cooperate**	2, 2	5, 0
	Defect	0, 5	4, 4

Table 2.2.2

A player who optimizes by choosing actions which minimize penalty is called a rational player. If both players are rational and both players know that the other player is also rational, then the players will optimize their own actions adjusted to the moves of the opponent player. This results in mutual defection *P*, which precisely is a Nash equilibrium [1]. This solution is a direct consequence of rationality. In the single-shot PD, to defect will always be the best solution.

2.2.1 Iterated Prisoner's Dilemma

Prisoner's Dilemma is used in discussion of economics, political science, evolutionary biology, and of course game theory. In many biological settings, the same two individuals may meet more than once. If an individual can recognize a previous interactant and remember some previous aspects of the prior outcomes, then the strategic situation becomes an *Iterated Prisoner's Dilemma (IPD)* with a much richer set of possibilities. [2] The players can still not communicate prior to their decisions, but it is assumed that each player can see what the opponent did in the previous round, and the round before that. This makes new strategies and tactics possible.

2.2.1.1 Fixed number of interactions

The IPD can be played with either a fixed number of interactions in advance or that there is some probability that after the current interaction the two players will meet again. With a known fixed number of interactions, the Nash equilibrium still stands. If both players are rational, mutual defection will always be the best action in the last round; after all, the last round is just a single-shot version of the game. The actions will not get any futuristic consequences. If both players are rational they know that the other player will defect at the last round, and will therefore defect at the second last round as well, and so on. This reasoning is called *Nash backward induction*[1]. Even though this yields the clear prediction that both players will defect in every round of the game, [4] claims that cooperation early on in the finitely repeated game can be introduced *with* Nash backward induction in place provided some uncertainty is injected into the game. [4] describes that "…because it seems possible that an instrumentally rational player may be able to exploit some idiosyncrasy on the part of the other player so as to achieve the cooperation outcome (and hence superior returns over the long run) by playing some strategy which does not defect in all plays."

2.2.1.2 Indefinitely number of rounds

Nash backward induction depends on the players knowing how many rounds of IPD they will play. But if the players do not know when the last play will occur, then the argument has no starting point. The players are unable to calculate backwards from a certain point, but instead they have to use different types of strategies which only look forwards. By playing the IPD indefinitely, cooperation is a rational outcome [9].

[1] The difference between *backward induction* and *Nash backward induction* turns on the use of common knowledge of rationality (CKR). The former does not require CKR whereas the latter does [3].

2.2.2 Tournament

Robert Axelrod, a political scientist, organized a public tournament where participants could submit a computer program to play the IPD. The players were to select D or C. And the winner of the tournament was the player that did best overall, against all the other players. We will here describe some of the well known players that took part in this tournament. These players are considered to be fixed structure automata (FSA) as described in chapter 2.1.4. Random and the two last players are exceptions since these players have no ability of learning or responding on the actions of the opponent.

Tit for Tat
The simple player Tit for Tat was elected as the winner in the first tournament. Tit for Tat starts with cooperation, and then repeats the opponent's moves in next round

Tit for Two Tat
Tit for Two Tat plays like Tit for Tat but forgives one defection. It defects after two consecutive defections.

Random
Random player makes random moves. It defects or cooperates with a probability of 0.5 in every move, unaffected of the moves of the opponent.

Grim
Grim Player starts with cooperation, but will always defect after one single defection from the opponent. This player is also known as Friedman.

Tester
Tester always starts with defection on the first move, and cooperates in the second move. It is testing the response from the opponent. If the opponent reciprocates the first defection, Tester will go on playing Tit for Tat. However if the opponent continues to cooperate, Tester will continue to alternate between cooperation and defection until the opponent defects, where it will continue with Tit for Tat strategy.

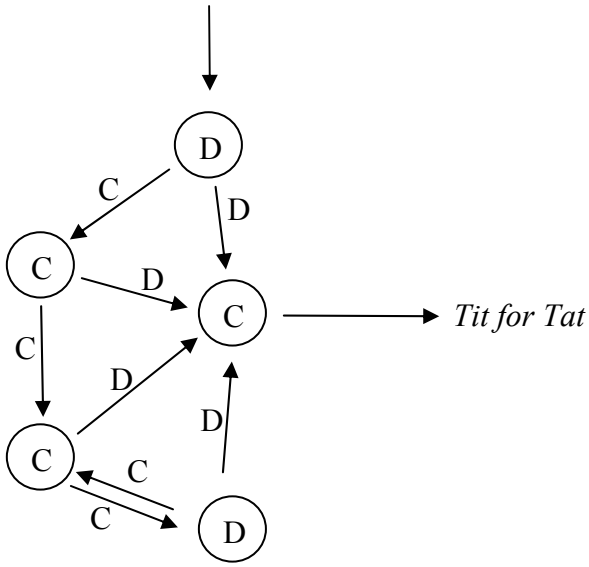

Figure 2.2.1

Joss
Joss starts with cooperation, and as Tit for Tat it always reciprocates a defection, but only reciprocates cooperation ninety percent of the time.

Cooperate
This player always plays cooperation

Defective
This player always plays defection

3 Interconnected Automata and Nonstationary environments

This chapter introduces theory around more advanced topics. The first topic deals with Interconnected Automata which expands the idea of learning automata even further by connecting them in a network. The second topic deals with nonstationary environment which expands the environment notion introduced in chapter 2.

3.1 *Interconnected Automata*

Humans are able to produce more if they work together. It is vital to have a well structured organization to benefit from all the participants. A company may be organized with a leader and several specialists with their own special skills in some areas. The leader might delegate assignments to the person that is best suited to solve a problem. Other companies may be organized differently. A factory may have some people to do different part of a production. There are different workers who work synchronous on different stages on the production of the same product. The same structures apply to automata. Narendra and Thathachar describe two models of interconnected automata, synchronous and sequential [6].

3.1.1 Synchronous Models

In these models, all automaton acts into their respective environments synchronously, while the response of the environment E_i is the input of A_{i+1}. The basic structure is shown if Figure 3.1.1

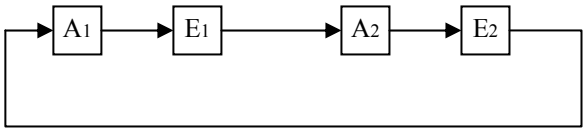

Figure 3.1.1

3.1.2 Sequential Models

While in synchronous models all automata acts at the same time instant, in sequential models only one automaton acts at any time instant. The action chosen decides which automaton to act next. A sequential model can appear as a tree structure or as a network, and different variants of both.

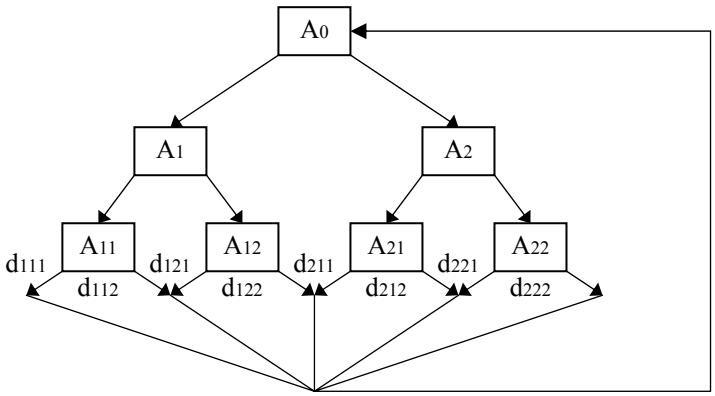

Figure 3.1.2

In the tree structure shown in Figure 3.1.2, A_0 decides which automaton to act, A_1 and A_2 decides which automaton is to make the actual decision. Every automaton in a path, updates its probabilities only after a complete path has been established. This means that A_2 will not update its probabilities until it is to make its next decision based on all the events in between.

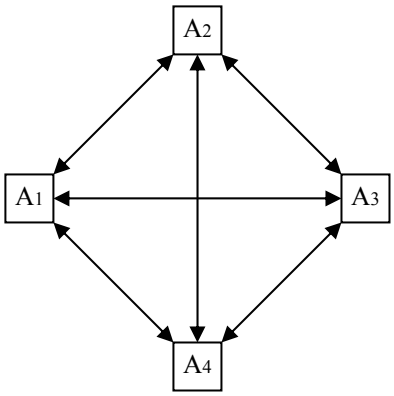

Figure 3.1.3

Figure 3.1.3 shows a general network of automata. In this network every automaton may choose all of the other automata to act next.

3.2 *Nonstationary Environment*

In a nonstationary environment, the probability of getting a penalty from the environment given an action will change over time. The environment may change; meaning an optimal action in one moment may not necessarily be the optimal action the next. In a nonstationary environment, the automaton should change its choices of preferred actions over time, because following the same strategy all the time will most likely not lead to an optimal solution. Learning in this case should therefore be able to track the optimal cases, and be flexible enough to change.

The question to be answered is: what does change involve? An environment is nonstationary if the penalty probabilities c_i (i = 1, 2,…, r) corresponding to the various actions vary with

time[6]. This change can be constant with changes to $c_i(n)$ by small increments or happen in discrete steps where the environment remains constant in an interval [n, n+T-1] and switch at n+T.

A simple example from an analytical point of view, is an environment that only change between a finite set of stationary environments E_i (i = 1, 2,…, d) according to a specific rule[6]. In this case one can imagine having a set of automatons A_i (i = 1, 2,…,d) where A_i is only updating its action probabilities when E_i is the current environment (A_i being a subset of automatons in a hierarchy). This approach has two problems. First, since every environment E_i will have its unique automaton A_i, d can not be large number practically. The larger d is the higher is the chance of having automatons A_i that never converge to an optimal action, since there are several automatons that will need to learn. Second, the automaton must be aware of which environment it is operating in. This is a requirement that often is not satisfied, because of the nature of an automaton working toward unknown environments.

Narendra and Thathachar [6] introduces two additional types of environments that are nonstationary. These are Markovian Switching Environment and State Dependent Nonstationary Environments. Markovian Switching Environment introduces the idea of defining environments as states of a Markovian chain. In this case the environments are connected to each other in a finite chain of states, where the input to the current environment (or state) will affect what will be the next environment (or state). This is the same way one can look upon fixed structure automata, where the automata themselves are finite set of states in a Markovian chain. In Markovian Switching Environment the set E_i (i = 1, 2,…, d) is now the states of this markovian chain. In State Dependent Nonstationary Environments the set $E = \{E_1, E_2, \ldots, E_d\}$ is the finite set of states the environment can take. The states vary with the stage number n.

If we look at learning automata connected in a hierarchy we indirectly get a nonstationary environment. If the hierarchy for instance contains two levels of automata where one automaton is connected to two sub-automata, the environment from the top-level automaton's point of view will be nonstationary. In this case, the actions of the top-level automaton are the choice of which sub-automata to use. The chosen sub-automaton will in turn choose its action based on its own action probability distribution. The action performed will result in a response from the environment and the sub-automaton will update its action probabilities when the response is received. Because the lower-level automata change their action probabilities, the penalty probabilities related to the actions of the higher level automata will also change. Figure 3.2.1 is an example of such an automaton. The top-level automaton has two actions α_1 and α_2 which result in the execution of A_1 and A_2 in the same order. Each sub-automaton has two actions denoted by α_{ij}, where i symbolize the top-automaton and j the current sub-automaton.

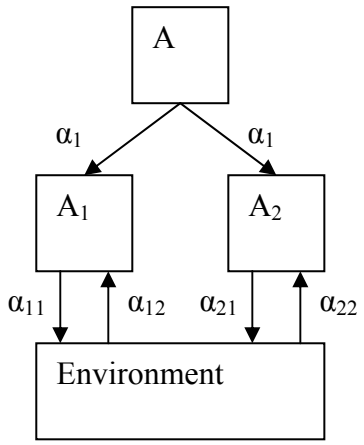

Figure 3.2.1

We also obtain a nonstationary environment in cases where the overall environment consists of several different environments connected in parallel, as seen in Figure 3.2.2

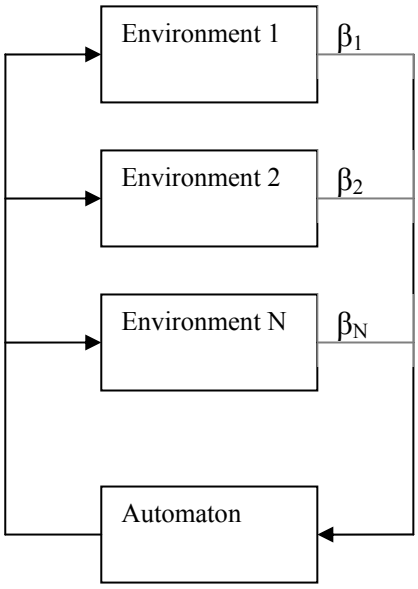

Figure 3.2.2

The automaton is connected to N environments denoted by E_1, E_2,…, E_N where each environment E_i (i = 1, 2, …, N) consists of an action set $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, an output set $\beta_i = \{0, 1\}$ and a set of penalty probabilities $\{c_1^i, c_2^i, \ldots, c_r^i\}$ where
$c_j^i = Pr[\beta_i(n) = 1 | \alpha(n) = \alpha_j]$

The automaton has the same action set α as the environments. Every environment connected to the automaton will respond to an action performed by an automaton. In the case shown in Figure 3.2.2 the N parallel environments will respond to an action by the automaton by creating a n-vector whose elements consists of elements from output set $\beta = \{0, 1\}$.

In order to study the behaviour of an automaton in a nonstationary environment, we have defined the nonstationary environment as being a switch of opponent strategy. For instance the opponent may switch from purely cooperative to purely defective. We have also made a simplification that these switches appears in regular intervals, however this is for making the study more simple and is not a necessary requirement. As soon as the learning player has learned the different opponent strategies, the opponent switch should be noticed in a short time even if it is regular or not. The automatons do not know when strategy switch occur. Switching the opponent strategy will most likely change the penalty probability attached to an action and thus satisfies the definition of a nonstationary environment.

4 Method

The Iterated Prisoner's Dilemma is a simple two player game where the players have to cooperate with each other in order to achieve an optimal solution for both. Unfortunately they do no not know what the other player chooses to do. If a player wishes to play optimal against an opponent it must learn the strategy, if any, that the opponent uses.

Several experiments have been performed to analyze the effectiveness of strategies by having different strategies compete in a tournament. The most famous one is the tournament arranged by Axelrod. In this tournament the so-called Tit for Tat strategy proved to be the winner.

This chapter will present two hypotheses that will form the basis of the work done throughout this thesis. Our goal is to gain empirical evidence that strengthen the hypotheses defined. We will use the hypothetico-deductive method to achieve this. Each hypothesis is built up with one or more predictions. Our job is to evaluate each prediction by doing observations on tests that are performed. If observations are contrary to the predictions they are evidence weakening the validity of the hypothesis. If observations are in favour with the predictions they are evidence strengthening the validity of the hypothesis. Since the tests we will perform may produce results that are sometimes for and sometimes against the hypothesis we must look at the average result over several tests in order to draw our final conclusion. In order to get accurate average results the tests need to be performed as many times as possible.

4.1 *Hypothesis 1*

Playing effectively against one strategy may be difficult, but further challenges occur when the opponent chooses to change its strategy during play. We can contemplate it as a game of mistrust. The player may believe they both have "agreed" on what is best to do, but the opponent suddenly decides to do something else. Off course, the change may be both for the best and for the worst. More advanced strategies are needed to handle such events effectively. In our work we have defined the following hypothesis:

H1: Interconnected learning automata will play better in a nonstationary environment than existing automata.

Several terms from hypothesis H1 must be explained. Interconnected automata and nonstationary environment has already been explained and defined in chapter 3. Existing automata is a broad term. We have selected a few of the strategies being used in the experiment by Axelrod. Tit for Tat is a good choice here since it is the winner of the competition performed by Axelrod. Tit for Tat is a fixed structure automaton, where the probability of doing an action given a result from the environment does not change over time. Other fixed structured strategies have been selected as well. We will be performing some tests using the Reward Penalty player as defined by Kehagias [7]. This learning player has a unique feature; it favours cooperation which is the best outcome for both players of the

game. The last term we need to define is playing better. This term is closely connected to the prediction that sets the foundation of our evaluation of H1. If H1 is true then we will expect the following prediction to be true

P1-1: Interconnected learning automata will get an average better score than existing automata.

Since the Iterated Prisoner's Dilemma is a score-based game it is natural to compare the scores of the players. Score in this case refers to the penalties given to the players. The lower total penalty the better total score. By observing the average score of interconnected learning automata versus Tit for Tat and other strategies we can evaluate the hypothesis H1. If the average score of the interconnected learning automata is better than the other strategies, the hypothesis will increase in validity. If not, the hypothesis will decrease in validity.

4.2 *Hypothesis 2*

Several techniques may be used to make an interconnected automaton more efficient. In our thesis definition we have decided to look at one of them. This gives the fundament of our second hypothesis.

H2: Game history will improve the effectiveness of an interconnected automaton.

Game history involves remembering the previous moves done in the game. This way the automaton can analyze the actions being made by the opponent and as a result learn to choose the best moves to take. To evaluate the hypothesis H2 we have two predictions, which we can expect to be true. If H2 is true both predictions are true. The first prediction compares interconnected learning automata with history against other interconnected automata.

P2-1: The interconnected automaton with history gets a better score than other interconnected automata.

We are to define two interconnected automata in this thesis. One of them will be using history and one of them will not. By comparing these we can see whether game history is a factor of improving effectiveness. This prediction does not cover all possibilities of opponents. We must consider other alternatives as well, which sets the foundation of the second prediction.

P2-2: The interconnected automaton with history gets a better score than other strategies.

If we can not evaluate whether an interconnected automaton with history plays better than other strategies such as Tit for Tat, we can not say that hypothesis H2 is evaluated completely. Tests will be performed to validate these predictions. The overall results of these tests will form the basis of the validation of the hypothesis.

4.3 *Evaluating the hypotheses*

In order to perform the tests we define an interconnected automaton. In our thesis we have designed and implemented two automata based on two different ideas of solving the problem. The first player is the Hierarchical player, based on the idea that an automaton can learn to use the optimal specified strategy against different opponents. The second player is the SLASH player; based on the idea that remembering game history may help in learning the optimal play against any strategy. There is also a third automaton which is an extension of the SLASH player, called I-SLASH. This one is based on the SLASH player, but with ideas from the Hierarchical player attached to it. These players are described in detail in chapter 5.

We will develop a prototype in order to gain empirical evidence in H1 and H2. This implies that it must be capable of running simulations of the Iterated Prisoner's Dilemma game using the learning automata and other strategies. This prototype is described in detail in chapter 6.

Several tests must be performed to validate our hypothesis and the questions given in the Thesis definition. The tests must involve running several strategies in several different environments. Using nonstationary environment is our primary concern; however tests may be done in a stationary environment, since learning in a stationary environment is easier to analyze and can be used as a reference. The scores the players achieve is the important factor. The total score in the end is not the only important property. We can learn a lot from looking at the average score of multiple rounds as they change throughout the game. This way we can get a good look at how the players behave. This is especially useful when looking at the learning players.

5 Interconnected algorithms

This chapter introduces the three new players designed and implemented by us to work in a nonstationary environment playing the Iterated Prisoner's Dilemma. Chapter 5.1 introduces the Hierarchical player which tries to learn the optimal strategy of the list of strategies given to it. Chapter 5.2 introduces the SLASH player which adds history to the learning process. Chapter 5.3 introduces an expanded version of SLASH called I-SLASH. This player consists of several SLASH player connected in a hierarchy.

5.1 *Hierarchical player*

Playing the Iterated Prisoner's Dilemma with a single fixed strategy is always a possibility, but not necessarily the optimal solution. Having an optimal play in this case depends solely on the strategy of the opponent. Having a single fixed strategy will likely be even more inefficient if the opponent change strategy during play. The Hierarchical player is designed to handle these problems. What actions to take, meaning what strategy to choose, will change as the game goes on. The idea is to have a player that chooses an optimal strategy against one opponent strategy, but if necessary, change the optimal strategy if the strategy of the opponent should change. Ideally the Hierarchical player will always play optimal against any opponent strategy.

As the name implies, the Hierarchical player is organized in a hierarchy. At the top of the hierarchy is a learning automaton choosing between different strategies. The top player will be referenced as top player or as a coordinator, while the strategies the automaton chooses from will be referenced to as subplayers throughout the thesis.

Let us first have a general look at how the Hierarchical player will work. Details will be discussed later. Initially every strategy has an equal chance of being picked. The chosen strategy will then be used for a set amount of iterations playing against an opponent. Based on the strategy and the opponent actions the penalty will be calculated after having its set of runs. Whether the strategy did good or not is determined by taking an average of the total penalty given and comparing it to a penalty limit. Updates of the action probabilities will now take place using the reinforcement scheme chosen. In a Reward Penalty scheme the chosen strategy will be rewarded if its average penalty is below the penalty limit. Using the Reward Penalty scheme will result in an increase of the probability of choosing the current strategy and decrease the probability of choosing the other strategies. If the average penalty was a penalty, the Reward Penalty scheme would increase the probabilities of the other strategies and lower the probability of the current strategy. After the update of the action probabilities the chosen strategy has finished its set of runs. The coordinator will now choose a new strategy based on the new action probabilities. This has been a general introduction of the Hierarchical player. We can now take a closer look at some of the details.

The Hierarchical player has no limit in the amount of strategies to choose from. However, the more strategies the longer time to find the optimal one, since each strategy has to be tested. Each strategy has an initial probability of being chosen. The sum off all strategy

probabilities equals to one. If a strategy has a probability of 1.0 of being chosen the Hierarchical player has converged toward one strategy.

In order to update the action probabilities, one needs a reinforcement scheme. The Hierarchical player supports the Linear Reward Penalty, Linear Reward Inaction and Linear Inaction Penalty schemes. These are described in chapter 2.1.

In order to update the penalty probabilities, we have to decide what a penalty is. In the game of IPD this is not given since every response from the environment can be looked upon as a penalty. It is imperative to notice the difference between a penalty in the game of IPD and penalty in the theory of Learning Automata. A Learning Automata somehow needs to map the penalties from IPD over to rewards and penalties it can use as learning material. In other words, we must differ between good penalties and bad penalties in the game of IPD. A good penalty is a penalty that can be treated as a reward by the Learning Automata. A bad penalty is a penalty that can be treated as a punishment by the Learning Automata. Kehagias [7] had one solution to this problem. He penalizes S and P and rewards R and T, using the cost definition in Table 2.2.1. This is effective against opponents where mutual cooperation is achievable. What Kehagias has done is to specify a penalty limit between R and P. A penalty higher than the penalty limit will be treated as a punishment by the automaton and a penalty lower than the penalty limit will be treated as a reward. If we consider the penalty values given in Table 2.2.2, setting the penalty limit to 5 will result in all responses from the environment being treated as a reward, which is equal to the Reward Inaction scheme. If the penalty limit is zero all responses from the environment will be treated as punishments, which is equal to the Inaction Penalty scheme. We can take this solution one step further by making the rewards and penalties dependant on the distance from the penalty limit. The lower a penalty is from the penalty limit the higher reward to the automaton and vice versa. This is the solution used in both the Hierarchical player and the SLASH players. Choosing a suitable penalty limit in this case is of significant importance, since it will affect the overall behaviour of the players. A choice of penalty limit to the Hierarchical player can not be taken without considering another property of the player behaviour; the fact that it switch strategy during play.

Many strategies a Hierarchical player can undertake are dependent of the last move the opponent did. Also, the last move the opponent did is likely taken as a result of the last player's move. If the coordinator switches strategy, the new strategy is dependent on the move of the old strategy which is not necessarily ideal. As a simple example, consider a situation where the coordinator switches strategy from purely Defective to Tit for Tat. If the opponent is rational, it has learned to defect against the Defective subplayer, and it will then choose to defect on the next move after the switch. This will result in the subplayer Tit for Tat choosing defection the next time it will be chosen by the coordinator. The kindness of Tit for Tat may be ruined by the evilness of the purely Defective player, depending on the opponent strategy of course. In order to avoid such behaviour one has to take into consideration how often a coordinator can select a strategy. If strategy selections are allowed after all iterations, the learning will be inefficient or not present at all. Because of this, it is sensible to let each strategy play a number of times before a strategy selection is possible. However, each strategy should not be played too many iterations since this will slow down

the learning rate of the Hierarchy player. The idea is to update the action probabilities of the Hierarchical player after a strategy has been run a number of times. After a strategy has had its run, an average penalty of that strategy is calculated. This average will be used to determine whether the choice of behaviour is to be penalized or not. A number of tests have shown that running a strategy for 5 iterations before action probability being updated, results in a Hierarchical player that converges to the same strategy in several runs and also have a fairly quick learning rate. A coordinator that selects a strategy after every iterations, results in a Hierarchical player that converges totally random toward strategies. This means that the learned optimal strategy, if any, changes from each run.

The next challenge is to determine right from wrong, good from evil. What will be considered a punishment and what will be considered reward. Since the Hierarchical player updates its action probabilities after a strategy has had a few iterations, the penalty is calculated by the average penalty given during these iterations. If the average is above the penalty limit it is to be considered a penalty and if the average is beneath the penalty limit it is considered reward.
Deciding a penalty limit is not an easy task. It has been done using several tests where overall performance has been considered. The penalty limit influences the learning rate since it is calculated from how far the average score is from the penalty limit similar to the SLASH player. If the average score is close to the penalty limit, the learning rate will be a small number. If the average score is far from the penalty limit, the learning rate will be a high number. A series of tests have shown that setting a penalty limit to 2.5 yields a Hierarchical player that has a fairly quick learning rate.

5.2 *SLASH Player*

Stochastic Learning Automata with States of History (SLASH) is our suggestion of an efficient player, based on interconnected automata with use of history. The use of history makes it possible to outsmart simpler strategies, like some fixed structure automata. If a player is playing against Tit for Two Tat (described in chapter 2.2) the best strategy would be to alternate between cooperation and defection. A player with states of history can recognize this pattern. One state should always defect while another should always cooperate regarding to the previous moves.

SLASH consist of states, which represents the former move(s). The number of states depends on how many moves the history is supposed to remember. Each move consists of two parameters; one for the player and one for the opponent, for instance 'CD'. In this example, SLASH cooperated in its last play, and the opponent defected. The number of states will then be calculated with this formula:

*Number of states = (Number of moves * 2) ^ 2*

A history length of 2 will result in 16 states. The states are described like this: "CCDC". The first letter describes what SLASH did two moves ago. The Second letter describes what the opponent did two moves ago. The third and fourth letter describes the former move of the two players.

Each state has its own Linear Reward Penalty LA. To update the probabilities of the LA, one state evaluate an action when it is returned to the same state. The update formulas used are as follows:

Raise defection
$P(t+1) = P(t) + ((1 – P(t) * a)$

Raise cooperation
$P(t+1) = (1-a) * P(t)$

$a = á * \gamma$

The learning rate á which is usually set to 0.01 and the update factor γ depends on the output β from the environment. There are many reasonable ways to calculate γ. One solution is to use a penalty limit, as described in the previous chapter, which decides whether an action should be rewarded or penalized. The problem with this solution is how to define a perfect penalty limit.
After several tests we chose a Reward Inaction solution where every action is rewarded, but the volume of the reward depends of the amount of penalty given. This solution is developed simply by setting a penalty limit equals to the highest penalty (S) which will be 5, based on the cost values from Table 2.2.2.

To update the states probability the LA calculates the average penalty of each move until it is returned to its originating state. This is illustrated in the example below.

Example 1

1. State = 0
2. SLASH = defect, opponent = cooperate; Penalty = 0
3. State = 2
4. SLASH = cooperate, opponent = cooperate; Penalty = 2
5. State = 8
6. SLASH = cooperate, opponent = cooperate; Penalty = 2
7. State = 0

Figure 5.2.1 shows the states and the movement. SLASH starts in state 0. When it is back in state 0, the average penalty has been 4/3. To determine whether this was a profitable or unprofitable act, the state needs a penalty limit which describes the limit of what is considered a profitable act. The penalty limit may be a constant or it can vary. In this case the penalty limit equals to the highest penalty limit which is 5. This results in a Linear Reward Inaction algorithm. This example rewards defection from State 0 with 5 – (4 / 3) = 3.67.

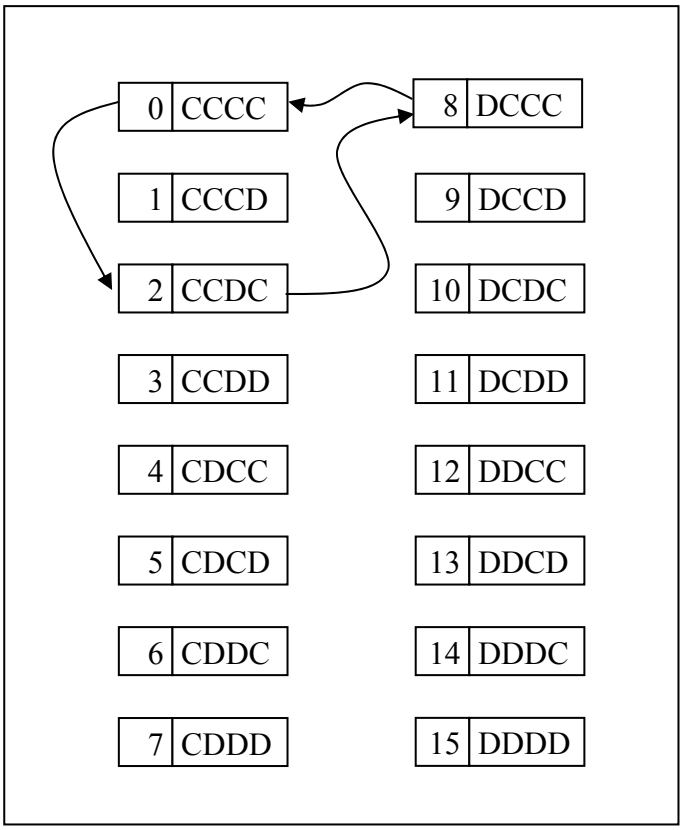

Figure 5.2.1

This concept is based on a sequential model and a general network, where one automaton (state) may lead to any of the other automata. There is neither one single automaton that controls the movement. This model is described in 3.1.2.

5.3 *I - SLASH player*

I-SLASH is a variety of SLASH which is an abbreviation for "Interconnected SLASH". I-SLASH consists of three SLASH players, but could theoretically consist of many more as well. It connects the SLASH players together in a hierarchy, with one 16-state SLASH player as the coordinator. The task of the coordinator is to predict the opponent move. If he predicts the opponent to cooperate, he will let the first automaton play, and if he predicts the opponent to defect he will let the other automaton play. If the coordinator predicts correct it will be rewarded and if it predicts wrong it will be punished. This learning is called supervised learning, since the environment has exact knowledge of what action is to be considered correct. This feedback is easier to deal with compared to the response from playing the IPD. The feedback from the IPD is graded with four different penalties and the player itself has to consider whether a feedback is a reward or a penalty.

With a correct response from the environment instead of a graded feedback, the coordinator in I-SLASH may increase its learning rate. This gives faster update frequencies and more accurate predictions as well, especially against fixed structure automata.

The two subplayers which are specialists on either a cooperative or defective opponent only have 4 states. The result remains accurate since the top-player compensates with its 16 states. Tests show that this solution is able to learn the same extend of complexity as a 16-state-SLASH player. Using 4 states are considerable faster to train, compared to 16 states and the coordinator has a faster learning rate than a traditional SLASH player. All things considered, this gives I-SLASH quicker learning compared to SLASH, and the result remains accurate.

I-SLASH is like SLASH organized as a sequential model, but while SLASH is a general network; I-SLASH is a tree structure like Figure 3.1.2.

6 Prototype

Several issues in the thesis definition require tests in order to be evaluated. A prototype fulfilling certain requirements needed to be made to perform the tests. This prototype has all the functions required to simulate the game of Iterated Prisoner's Dilemma in a nonstationary environment.

6.1 *Requirements*

The following requirements were set.

- It must be able to play the Iterated Prisoner's Dilemma using both single strategies and supporting a Hierarchical strategy.
- It must support several types of strategies, from fixed to learning strategies.
- It must support both fixed and nonstationary environment.
- It must support some kind of visualization of the end result of games being played.
- It must support different cost values for the Iterated Prisoner's Dilemma.

6.2 *Graphical User Interface*

The implementation was done using C# in the .NET environment. The resulting user interface is shown in Figure 6.2.1.

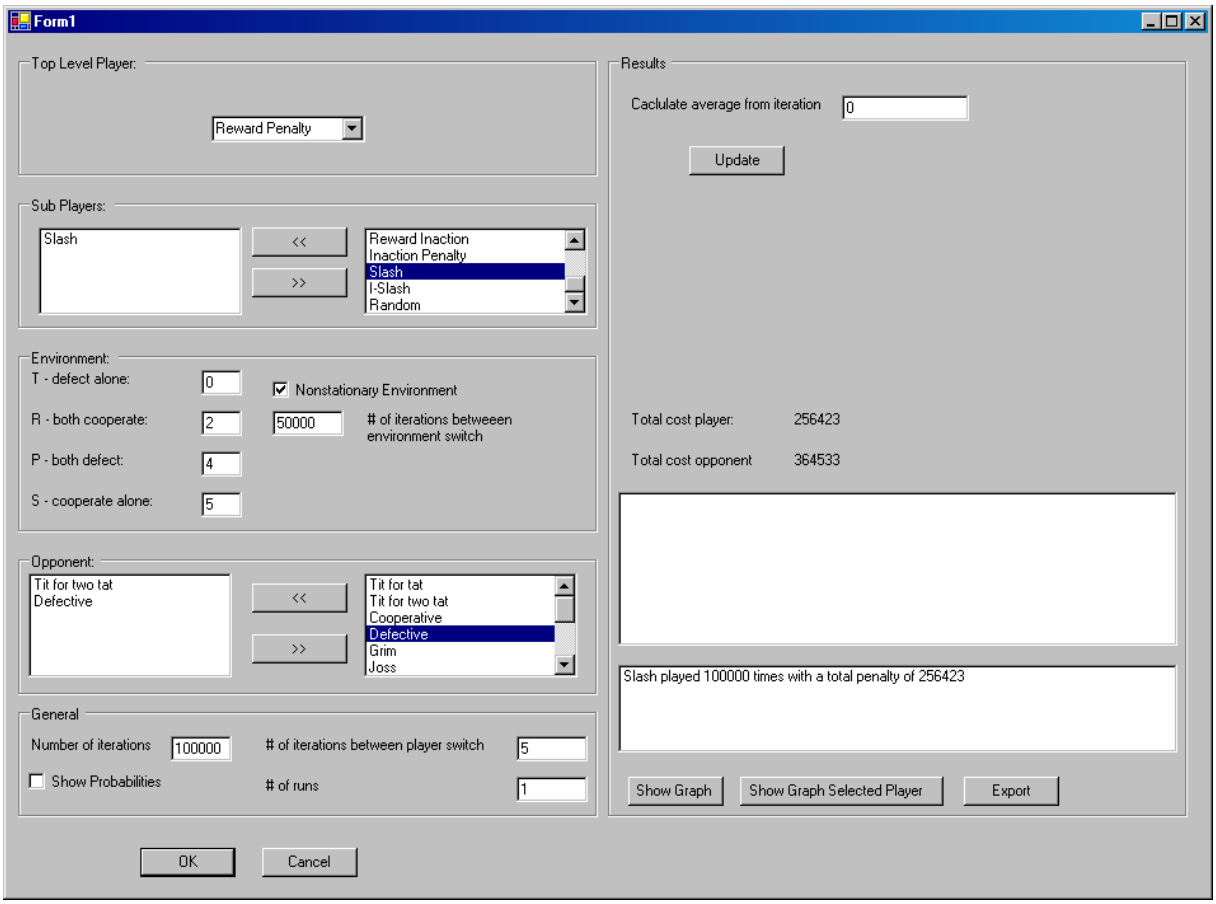

Figure 6.2.1

The prototype is built up around the Hierarchical player setup. This means there is a top player connecting several subplayers in a hierarchy. These are playing in an environment which is set up with different properties. Finally there is also an opponent set up with different strategies.

Let us look at the user interface which is designed toward the mentioned setup. At the top left portion of the interface, the frame called "Top Level Player", one selects the reinforcement scheme to be used for the top player. The supported reinforcement schemes are Reward Penalty, Reward Inaction and Inaction Penalty. The next frame, "Sub players", is where one chooses which strategies the Hierarchical player should be able to select from. The strategies supported are "Tit for Tat", "Tit for Two Tat", "Cooperative", "Defective", "Grim", "Joss", "Tester", "Reward Penalty", "Reward Inaction", "SLASH" and "I-SLASH". The Reward Penalty, Reward Inaction and Inaction Penalty schemes are defined as by Kehagias [7].

The top player and the subplayers form the Hierarchical player. Organizing the design this way makes it possible to use both a single strategy and a Hierarchical strategy. If only one strategy is selected the Hierarchical player will choose this with a probability of 1.0 and is therefore equivalent to a single strategy.

The next frame is called "Environment" where the user is able to set certain environment specific variables, such as the penalties to be used and whether or not a nonstationary environment should be supported. The next portion is the "Opponent" frame where the opponent strategies are set. The strategies supported here are the same as in the "Sub players" frame. More than one opponent is possible to select. If the user has selected to use a nonstationary environment, the opponent will switch strategy between the strategies selected here. The last portion of the left side is the "General" frame which add some general options, such as the number of iterations the game should be having and also how many times a game should be run. This is also where the user selects how many times the Hierarchical player must run a chosen strategy until it is allowed to switch. The right side of the user interface involves the result of a game. The first textbox is used if the user wants to calculate the final score from a chosen iteration and to the end. The next portion is the total score. If a game has been run over several rounds the average score of all the rounds is displayed here. The following listbox is used to display the probabilities of the subplayers being selected by the Hierarchical player as they develop throughout the iterations. This is only listed if "Show Probabilities" is selected under the "General" frame. Only the probabilities of the last round are shown. The next listbox gives a short summary of the last game being played. The button "Show Graph" displays the subplayer probabilities of the last game over time. The button "Show Graph Selected Player" gives the user a possibility of selecting one of the subplayers and showing a graph of its behaviour over time. The "Export" button is the best way of displaying the results of the game. It stores the average score over all the rounds in a semicolon separated text file. This file can easily be opened and manipulated with spreadsheet software like Excel. The last two buttons to be mentioned are the "OK" and "Cancel" button. The "OK" button starts a game while "Cancel" closes the program.

7 Test Results

Several tests have been performed to analyze the effectiveness of different players in a nonstationary environment and also to analyze the behaviour of our players. The results are presented in two chapters. Chapter 7.1 focuses on the tests regarding effectiveness of different players in a competition. Chapter 7.2 focuses on the behaviour of our interconnected automata. This chapter only introduces the tests and the results. Discussion around the results will appear in chapter 8.

7.1 *Total score*

To analyze the effectiveness of different players in a nonstationary environment we can look at the total score, or in this case, the total penalty the different players can gain. To get a notion of this, several tests have been performed where players have played 10 000 iterations against strategy switching opponents. The tests are organized around a competition where the interconnected automata and other strategies compete against these opponents.

Let us first look at what players to choose in our tests. Axelrod has previously performed tests similar to ours, but in a different setting. In the famous tournaments made by Axelrod, Tit for Tat was the player that gave the overall best result. Due to these results it is natural to see how well Tit for Tat performs in our setting. Other strategies from the tournament will also be considered to give a broader view; these are Tit for Two Tat, Grim, Joss and Tester. These strategies are defined in chapter 2.2. Since we want to validate the interconnected automata, these will become an important part of the tests. SLASH and I-SLASH is therefore obvious players in our tests. The Hierarchical player must be validated with different subplayers, since each set of subplayers yields one unique Hierarchical player. In these tests we have five instances of the Hierarchical player with subplayers defined to be:

- Tit for Tat, Tit for Two Tat, Reward Penalty, Reward Inaction, SLASH
- Tit for Tat, Tit for Two Tat, Tester, Grim, Random
- Tit for Tat, Joss, Tester
- Tit for Tat, Tit for Two Tat, Cooperative
- Tit for Tat, Tit for Two Tat, Defective

In our test we will also use the Reward Penalty player as defined in [7].

The opponents form the nonstationary environment the players are to compete in. This is done by having the opponent switch strategy during play. In our tests the opponent will switch strategy thirty times during the 10 000 iterations. These switches will occur in regular intervals, meaning a switch will occur after a strategy has played 333 iterations. The switches could occur more random though, but are regular in order for the analysis to be easier. The point is the players do not know which opponent strategy they play against; neither do they know when a strategy switch occurs. Each opponent in the tests switches between two strategies, with one exception explained later.

We can now look at the setup of opponents. The opponent strategies have been selected on the criteria that they should not be too equal. In this we put the meaning that there must be some difference in the behaviour in the strategies used. An example of too equal strategies is Tit for Tat and Tester, where Tester will act exactly like Tit for Tat if it is given a single defection against itself. The opponents are as follows

- Tit for Two Tat, Random
- Joss, Grim
- Tit for Tat, Defective
- SLASH

In three of the cases we have a pair of strategies that will be switched thirty times during play. In the fourth case we have the SLASH player alone. Using SLASH as an opponent forms a nonstationary environment since its penalty probabilities will change during play. This is in accordance with what is described in chapter 3.2.

Finally, let us look at the optimal strategy one can achieve against the different opponent strategies. This is important since we want the players to play as close to the optimal strategy as possible. The optimal strategy against each of the opponents varies in each of the four cases mentioned above. The first case is Tit for Two Tat and Random. In this case the optimal strategy is to alternate between cooperation and defection against Tit for Two Tat, and to defect against Random. The second case is Joss and Grim where cooperation is the ideal strategy throughout the game. However, this is a difficult task to reach since Grim will not forgive a single defection. The third case is Tit for Tat and Defective which yield an optimal strategy of cooperation against Tit for Tat and defection against the Defective player. In the fourth case, SLASH, the wisest thing to do is probably to cooperate. However, it may be the case that a player may fool SLASH through its use of history.

Game 1, Opponents: Tit for Two Tat, Random

The results obtained by the different players playing against Tit for Two Tat and Random are shown in Figure 7.1.1.

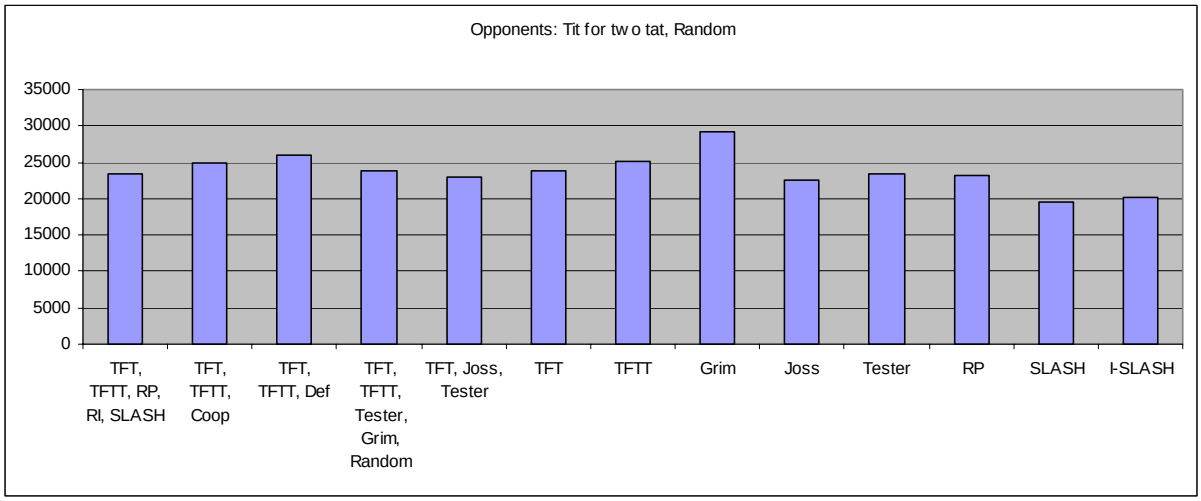

Figure 7.1.1

The figure shows a column for each of the players in this test. The different players are listed along the x-axis. The first five columns are the different instances of the Hierarchical player. The score is listed along the y-axis.
As seen in the figure, the SLASH players get the best result. Grim however does the worst play. It can be noted that the Hierarchical player does not get a very good score in any of the cases.

Game 2, Opponents: Joss, Grim
Playing against Joss and Grim makes Tit for Two Tat the supreme winner with a score of 21 326, whereas the worst player is yet again Grim with a score of 39 953. Other notable static players are Tit for Tat with a score of 37 824 and Joss with a score of 39 457. The SLASH players are among the best players, but with a score of 34 760 to SLASH and 34 137 to I-SLASH they are doing far worse than Tit for Two Tat. The only player to do better than SLASH and I-SLASH other than Tit for Two Tat is the Hierarchical player with Tit for Tat and Tit for Two Tat and cooperative as subplayers. This player is given a penalty of 31 562.

Game 3, Opponents: Tit for Tat, Defective
The third game contains the opponents Tit for Tat and Defective. The results are shown in Figure 7.1.2.

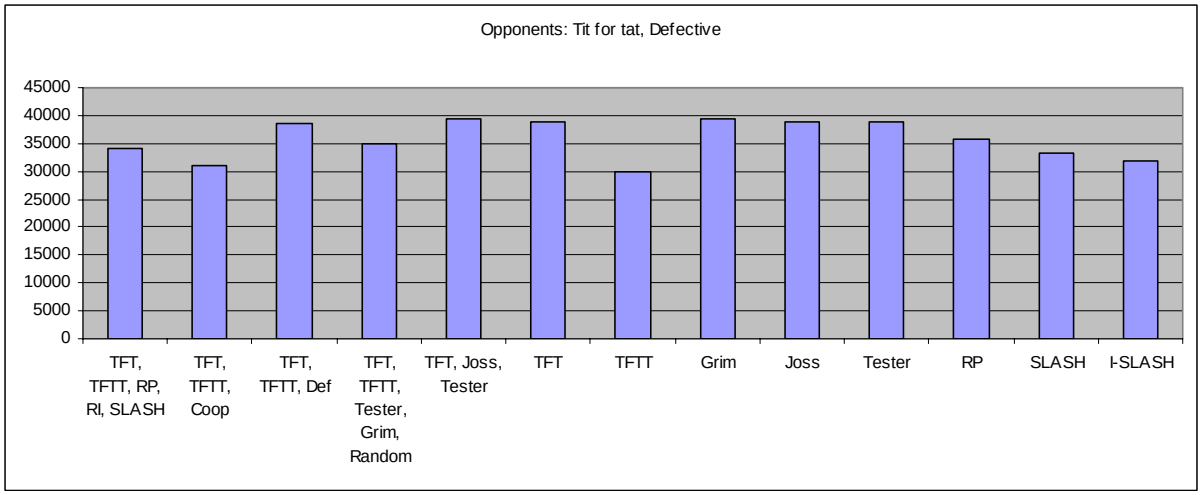

Figure 7.1.2

Once again it is noted that Tit for Two Tat is the winner. SLASH and I-SLASH are doing very well, but in close competition with the Hierarchical player using Tit for Tat, Tit for Two Tat and Cooperative as subplayers. The losers of this game are Tit for Tat, Joss, Tester and Grim. Also the Hierarchical player using the subplayers Tit for Tat, Tit for Two Tat and Defective and the subplayers Tit for Tat, Joss and Tester is doing badly.

Game 4, Opponent: SLASH
The last game is played using the SLASH player as opponent. As already mentioned this automatically gives a nonstationary environment. Tit for Two Tat is among the worst players in this round with a score of 34 090 only beaten by Grim with the score 37 315 and Reward

Penalty player with the score 35 531. Tit for Tat, Joss and Tester does very well in this case with a score of 21 795, 21 620 and 21 535. Playing against itself SLASH get the score 21 144 and is therefore the winner of this round, I-SLASH however does not manage this case well at all and is given a penalty of 27 050. The Hierarchical player does mostly well in this scenario, especially using the subplayers Tit for Tat, Joss and Tester which gives the score 23 248.

7.2 *Converging and behaviour*

In this section we will show the behaviour of SLASH and the Hierarchical player. We will show how they converge and how effectively they adapt after a change in the environment. The first experiment compares all the players described in 2.2.2 with SLASH, the variant I-SLASH and a Hierarchical player as well. The result shows the average penalty over 200 000 iterations. The exception is the last column where the opponent consists of three players. Here the result shows the average over 240 000 iterations. Some of the opponents have two or three players. In these cases the players are switched after 20 000 iterations. The values in Table 7.2.1 are the average penalty over tests performed 50 times to get an accurate result. The first row is the optimal play against each opponent. Inaccuracy is represented with '*' in the calculation of the optimal strategy. These opponents have a certain degree of random behaviour and are therefore unpredictable. The best score performed by a player is emphasized in **bold**. The rows represent the players, while the columns represent the opponents. The score listed are the average penalty gained by the players against the corresponding opponents.

	TFT	TFTT	Grim	Tester	Joss	Random	Coop	Def	TFT / TFTT	Joss/ Grim	TFT/ Def	TFT/ TFTT/ Coop	Avg
Optimal	2.00	1.00	2.00	2.00	2.3*	2.00*	0.00	4.00	1.50	2.15*	3.00	1.00	1.85
I-Slash	2.07	1.12	4.02	2.07	2.36	2.07	0.07	4.02	**1.67**	3.24	3.06	**1.19**	**2.26**
Slash	2.07	1.10	4.02	2.07	2.38	2.37	0.10	4.02	**1.67**	3.24	3.07	1.21	2.29
TFT	**2.00**	2.00	**2.00**	**2.00**	2.50	2.75	2.00	**4.00**	2.00	2.75	3.22	2.00	2.44
Tester	**2.00**	**1.00**	4.00	**2.00**	2.50	2.75	1.00	**4.00**	2.00	2.83	3.23	2.00	2.44
TFT/ TFTT/ Def	**2.00**	2.00	4.00	**2.00**	2.35	2.07	**0.00**	4.00	2.00	3.62	3.62	2.00	2.47
TFTT	**2.00**	2.00	**2.00**	3.50	**2.27**	3.12	2.00	**4.00**	2.00	**2.14**	**3.00**	2.00	2.50
Coop	**2.00**	2.00	**2.00**	3.50	**2.27**	3.50	2.00	5.00	2.00	**2.14**	3.50	2.00	2.66
Joss	2.50	1.82	4.00	2.50	2.56	2.71	1.18	**4.00**	2.21	3.66	3.61	2.07	2.74
Grim	**2.00**	2.00	**2.00**	4.00	4.00	**2.00**	2.00	**4.00**	2.00	4.00	3.90	2.00	2.83
Random	2.75	1.88	4.50	2.75	2.83	2.75	1.00	4.50	2.31	3.67	3.63	1.88	2.87
Def	4.00	4.00	4.00	4.00	4.00	**2.00**	**0.00**	**4.00**	4.00	4.00	4.00	2.67	3.39

Table 7.2.1

As we can see, all the fixed structure automata (FSA) used in this test as the opponent, have one or more corresponding FSA to play the optimal strategy. However against some of the

switching opponents, none of the players are able to play optimal. SLASH and I-SLASH are not able to play optimal against any opponents but are however usually close to optimal. I-SLASH and SLASH are also the players with the lowest average penalty.

7.2.1 Switching in a nonstationary environment

A nonstationary environment may change over time. To test the ability of the players in a nonstationary environment we have done several tests where the opponent changes its strategy. Figure 7.2.1shows I-SLASH playing against Tit for Tat, Defective. The test is performed 50 times and the graph shows the average penalty. The figure shows that the average penalty improves.

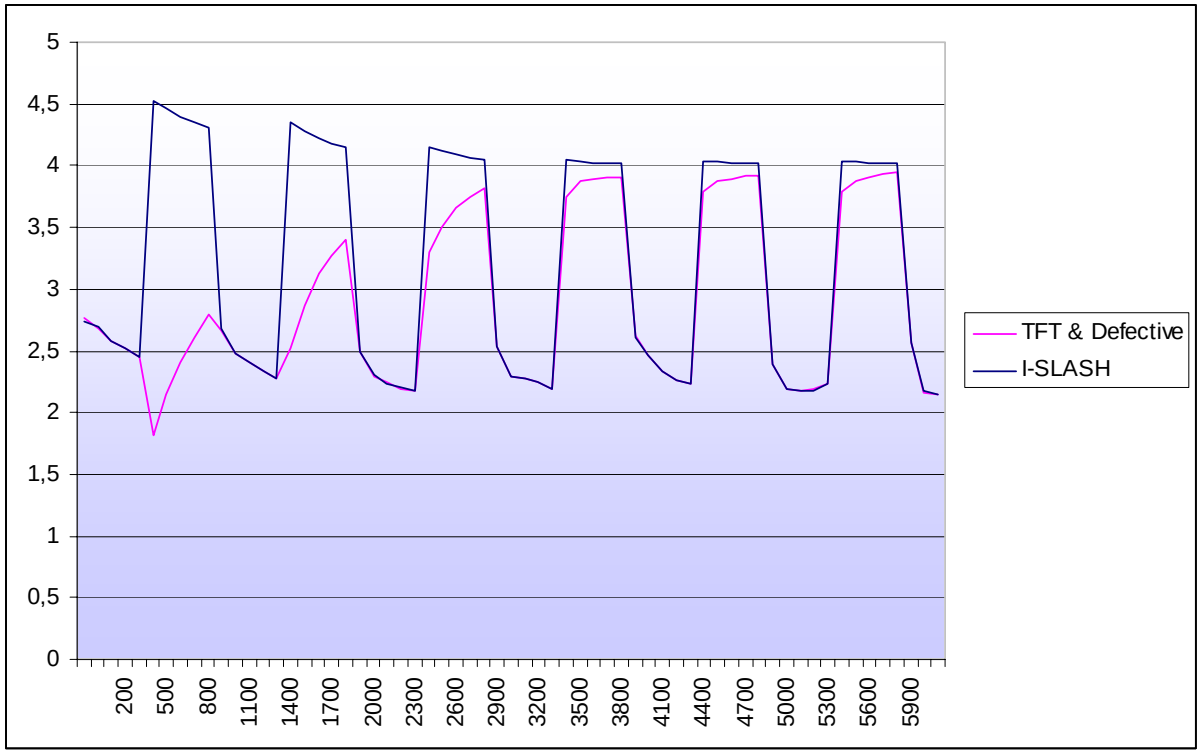

Figure 7.2.1

To measure the exact cost of a switch after the player has learned to play in a switching environment, we have performed a test on two switches as described in Figure 7.2.2. The game is still I-SLASH versus Tit for Tat, Defective. The figure shows results after 160 000 iterations. At this time I-SLASH has learned to play well against both opponents. The test measures the average penalty over 40 000 iterations. It starts with a switch and the opponent switches again after 20 000 iterations.

If we assume there is no cost in a switch, we are able to calculate the expected penalty based on values from Table 7.2.1; I-SLASH gets an average penalty of $i = 2.07$ against Tit for Tat, and an average penalty of $j = 4.02$ against Defective. Without any cost of a switch the assumed average $avg_{ij} = 3.045$. Instead the average penalty $avg = 3.052$. This is a total increase in penalty of 280 on these two switches, which is an average increase of 0.007 per

round. These two switches are described graphically in Figure 7.2.2. In a closer study of the numbers, it is clearly that most of the increase in penalty is located after the switch from Defective to Tit for Tat.

The total cost of a switch in an environment will change depending on the opponents. For instance, if the optimal play against both opponents will cause the same actions from the opponents. With Tit for Tat and Cooperative as opponents, the cost under the same circumstances as the test above, will result in a total penalty of 2 475 which is an average of 0.062 per round.

As a reference Tit for Tat will not gain penalty on the switch itself, since it switches immediately, however it will not play optimal against any of these environments. The overall result is inferior to I-SLASH.

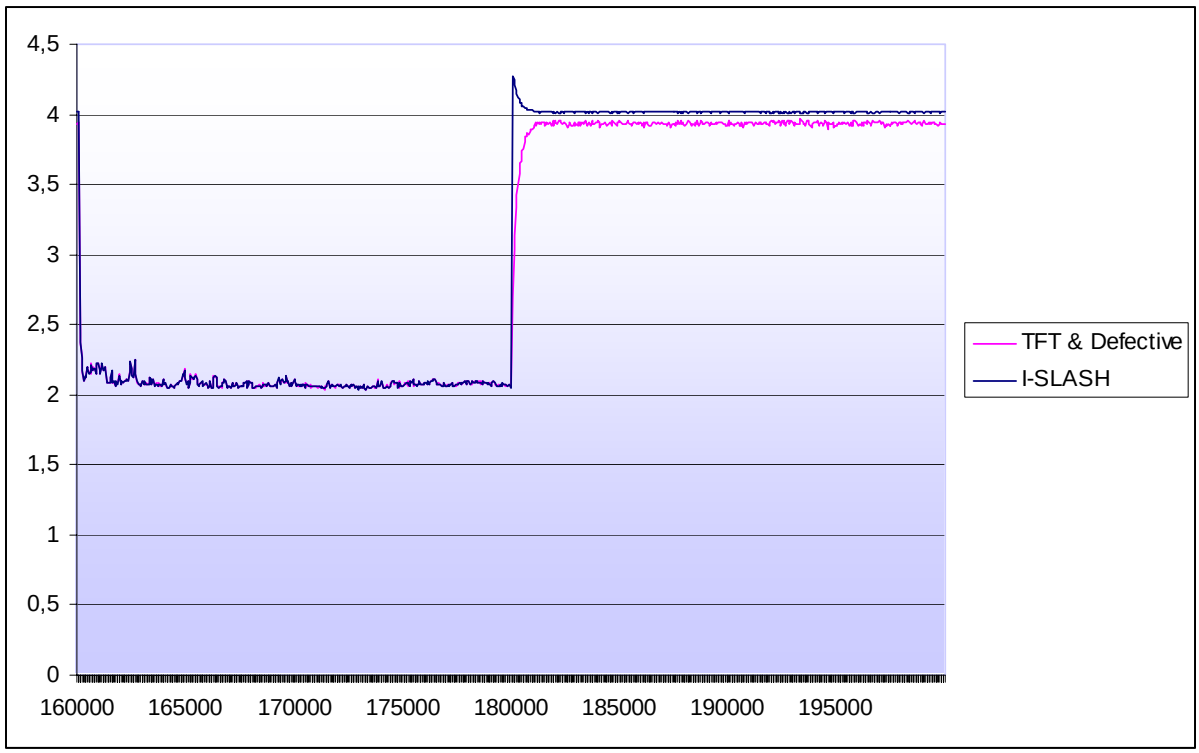

Figure 7.2.2

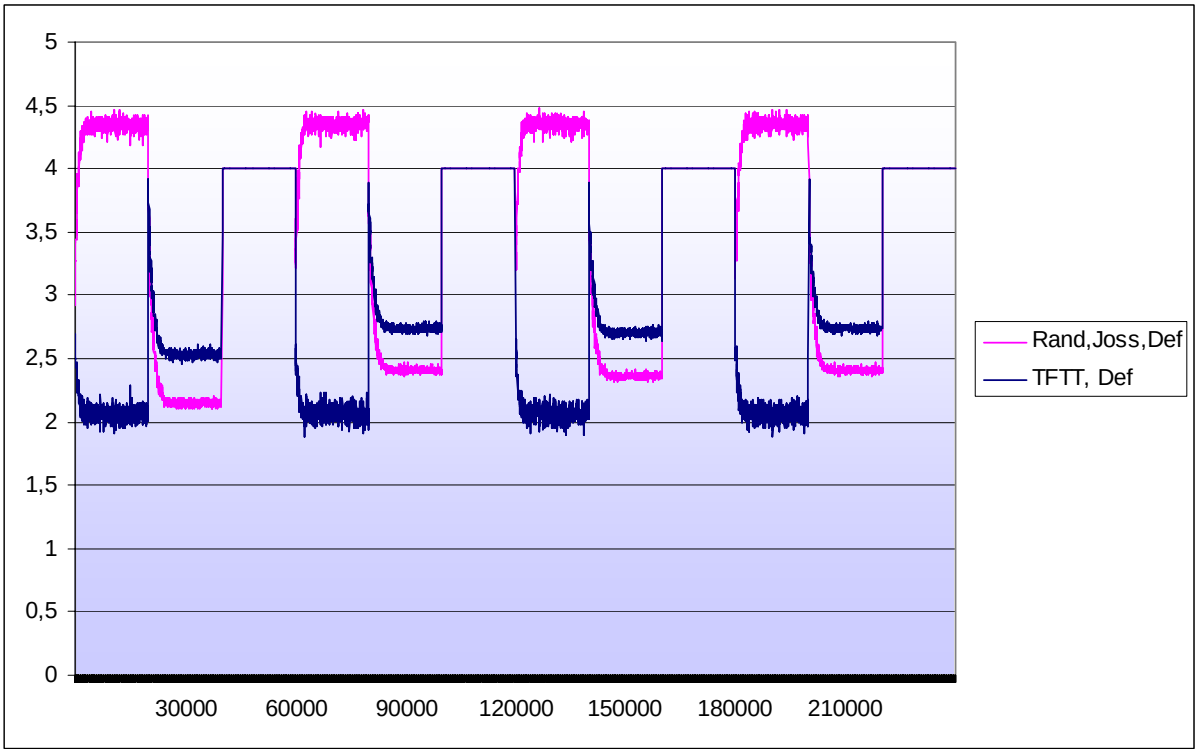

Figure 7.2.3

Figure 7.2.3 shows the Hierarchical player playing in a nonstationary environment against Random, Joss and Defective. In this game, Tit for Two Tat and Defective are used as subplayers. This game is played over 240 000 iterations. The player faces each opponent four times. Based on values from Table 7.2.1; the subplayer Defective gets an average penalty of $i = 2$ against Random, the subplayer Tit for Two Tat gets an average penalty $j = 2.27$ against Joss, and the subplayer Defective gets an average penalty of $k = 4$ against the defective opponent strategy. All these results are optimal play. The expected average will therefore be $avg_{ijk} = 2.757$. Instead the average $avg = 2.942$. As a reference to the Hierarchical player, I-SLASH manages to get 2.899, with the expected average of 2.817. The Hierarchical player is able to play optimal only against defective in this experiment. Close study of the results also shows that the switch towards a Defective strategy is performed almost immediately. The Hierarchical player needs more time to adjust itself after the switches to other strategies.

8 Discussion

The test results presented in chapter 7 have been analyzed thorough. Fixed strategies such as Tit for Tat does not need time to learn, thus having built-in responses to the opponent moves. When a player needs to learn the opponent tactics and base its actions upon experience, complexity arises. The Hierarchical player has a slow learning rate and is also always limited by its set of strategies to choose from. However, given the right set of actions the Hierarchical player will do well. SLASH and I-SLASH performs very well. They manage to learn to play close to optimal against most strategies.

Chapter 8.1 discusses the results presented in chapter 7.1 and chapter 8.2 discusses the results presented in chapter 7.2.

8.1 *Total Score*

We will now study the results of the nonstationary effectiveness tests presented in chapter 7.1 in detail. In this test we played the game for 10 000 iterations and having 30 switches of opponent strategies. This is a lot of switches in a short time. In fact, a learning automaton only has 10 000 / 30 = 333 iterations to adapt to the new strategy. The Hierarchical player has a disadvantage here. The hierarchy consists of two or more subplayers that must be tested and evaluated in the environment. As we remember from chapter 5.1 every subplayer are given five iterations in which they will be evaluated from. This gives an amount of 333/5 = 66.6 iterations of effective learning. In other words, the Hierarchical player has a short amount of time to learn. In the analysis conducted further we will first look at the single players and then look at the interconnected players to compare how well they play.

Game 1, Opponents: Tit for Two Tat, Random

This setup of opponents is interesting since both are special in what is the optimal play. At first sight the optimal play against Tit for Two Tat is purely cooperating, but this is not the case. Since Tit for Two Tat forgives one defection the optimal strategy would be to use this property by alternating between cooperation and defection. Learning this is difficult, so most players will have to use the second best strategy which is purely cooperating. Random is a difficult player to learn, since there is no apparent logic in its choice of behaviour, which is also the truth. Since there is no way of predicting what choice of action Random will use, the most obvious choices of strategy is to defect since this is an action that will never be punished. A nice thing with this choice of opponents is that if the player should choose to defect against the Random opponent, the Tit for Two Tat opponent will forgive this defection when the opponent switches strategy again. Tit for Two Tat will forgive one defection after the switch from Random. This gives the player a chance of learning to cooperate against Tit for Two Tat again, unless the player has the ability to learn to alternate between cooperation and defection. If the opponent was Tit for Tat instead of Tit for Two Tat this would not be the case.

Of the single strategies, Joss gets the best score. Joss has a great advantage against Tit for Two Tat because of its tendency to choose defection 10% of the time. Tit for Two Tat will

forgive this treachery and get the suckers payoff. Defecting like this also gives Joss a favour against the Random opponent that a Tit for Tat player would not have. Tit for Tat will get hurt since it will copy the behaviour of Random, but still do good overall. Joss will not copy Random like this. Its 10% defection rate against Random gives it an advantage since either if Random did cooperate or defect the result would be better than a suckers payoff. Tester and Tit for Tat will play this scenario almost identically. The biggest difference is the start where Tester will do a couple of defections until it is punished for it. The uncertainty with Random will also make sure that the players don't get identical score. Grim is the overall worst player. The reason for this is its unforgivable nature. A defection from the Random opponent is inevitably which Grim will never forgive. It will turn to pure defection which will be punished again by the Tit for Two Tat player.

Of the interconnected automata the SLASH player and its variant I-SLASH plays the game best. In fact, they win the game overall. The reason for this is that SLASH has the ability to learn the optimal strategy against Tit for Two Tat. Against a Random opponent the SLASH players will learn to defect. In this case they play close to optimal against both opponent strategies. However, they will need some time to learn this. The Hierarchical player plays well this game, but suffers from the fact that it never manages to converge to a single choice of strategy. In order to manage this it needs more iterations to learn. Because of this, every strategy a Hierarchical player can put to use will have a chance of being used. We note that the Hierarchical player with Tit for Tat, Joss and Tester as subplayers is the winning combinations. Tit for Tat, Joss and Tester is also the optimal single strategies. The reason this combination does well, however, is the fact that the overall Hierarchical practically becomes a Tit for Tat player. If we look at the scores this combination comes close to the Tit for Tat player. The major difference is caused by the change of subplayer strategy against the Random opponent which will cause the Hierarchical player to not copy the behaviour of the Random opponent exactly like Tit for Tat does. Using the subplayers Tit for Tat, Tit for Two Tat combined with either a purely cooperative or a purely defective subplayer results in almost identical score. However the combination with pure cooperative does slightly better than the purely defective combination. Having a purely defective player as a subplayer is not good against Tit for Two Tat. This is because each subplayer strategy is being played for 5 iterations which will result in several iterations with both player and opponent defecting. Also, there will be some iterations resulting in suckers payoff when strategy switches from purely defective to a Tit for Tat or Tit for Two Tat strategy. Having a purely cooperating player as a subplayer is good against the Tit for Two Tat opponent, but will suffer against the Random player where it is likely to get several suckers payoff.

Game 2: Joss, Grim

This is another setup of interesting opponents. In this case they both share a property that is difficult to learn. In the case of Joss it will defect 10% of the times, but this defection is impossible to predict since it is totally random. Grim will not tolerate a single defection and will punish such treachery with going purely defective. Learning the behaviour of Grim is impossible since learning would involve defecting against the opponent to see the result of the action. This gives a significant disadvantage to the stochastic learning players; the best they can do is to learn the optimal action in the situation presented to them, mutual defection. Overall, the best outcome is resulted by cooperating against both opponents.

Let us take a closer look at the result of the tests. We can see how this is a difficult combination of opponents by looking at the average score of all the players in the test, which is 35 641. Ideally the score would be around 20 000 if both the player and the opponent cooperated over all 10 000 iterations. The expected ideal score is slightly more than 20 000 because of Joss's tendency of defecting. From the test results, Tit for Two Tat is the only player to achieve this with a score of 21 326. It is actually superior since the second best player, the Hierarchical player with Tit for Tat, Tit for Two Tat and Cooperative as subplayers, achieves a score of over 31 000. Tit for Two Tat does well because of its forgiving nature which is excellent against both opponents. First of all it will forgive the defections done by Joss, unless Joss should decide to do two defections after each other. This case is unlikely. Secondly, it will cooperate against Grim. Tit for Tat is doing much worse. The reason for this is the fault of Joss. When Joss does its occasional defection, Tit for Tat will retaliate by defecting in the next iteration. Since Joss reacts to defection the same way as Tit for Tat, it will notice this defection and retaliate on the next iteration. The result is Joss and Tit for Tat alternating between cooperation and defection as show in Table 8.1.1.

	1	2	3	4	5	6	7	8	9	10	11	12
TFT	C	C	C	C	C	D	C	D	C	D	C	D
Joss	C	C	C	C	D	C	D	C	D	C	D	C

Table 8.1.1

Behaving like this gives a 50 % chance of defecting against Grim after the opponent switch. If this happens it will make the score even worse for Tit for Tat; one defection against Grim and Grim will never cooperate again.

Using Grim as a player gives the worst score, since it will choose to defect the rest of the game, after Joss decides to do a single defection. Joss as a player is also doing badly against these opponent strategies. First and foremost it plays mostly like Tit for Tat which is not good. Secondly it will do matters even worse for itself by doing a defection against the Grim player. Tester will also do badly in this case, since it will eventually end up playing as Tit for Tat with all the disadvantages mentioned earlier.

Looking at the instances of the Hierarchical player, the instance using Tit for Tat, Tit for Two Tat and Cooperative is the one rewarded with the best score. The reason for this is simply the fact that there is little chance of any of these subplayers to defect. The Tit for Tat subplayer may arrive in the same situation as mentioned above, with Tit for Tat switching between cooperating and defection against Joss. However, this situation may be resolved by the Hierarchical player by choosing the Tit for Two Tat or the Cooperative subplayer. These subplayers will make Joss go back to cooperating. The chance of doing a defection at the switch of opponents causing Grim to defect, is even less. Replacing the cooperating subplayer with a purely defective player will make the result worse. When the defective player gets its turn of play it will cause Joss to defect making it hard for the other subplayers to play ideally. The Defective player will also ruin the Grim opponent as soon as it gets a chance of playing against it. This combination of subplayers gets a score of 39 237. This is

slightly better than using Tit for Tat, Joss and Tester as subplayers which results in a score of 39 604. This instance of the Hierarchical player is built up with the single strategies that do weak by themselves. Since all of these subplayers almost behave as Tit for Tat, the disadvantages mentioned is existing here too.

Of the stochastic learners, the SLASH player and its variant I-SLASH have the best results. SLASH will not manage to keep Grim cooperating, but will learn to defect against this player. It will also cooperate against the Joss player. Figure 8.1.1 shows this. The figure displays average score of the players over 10 000 iterations. Score is listed along the y-axis and iterations along the x-axis.

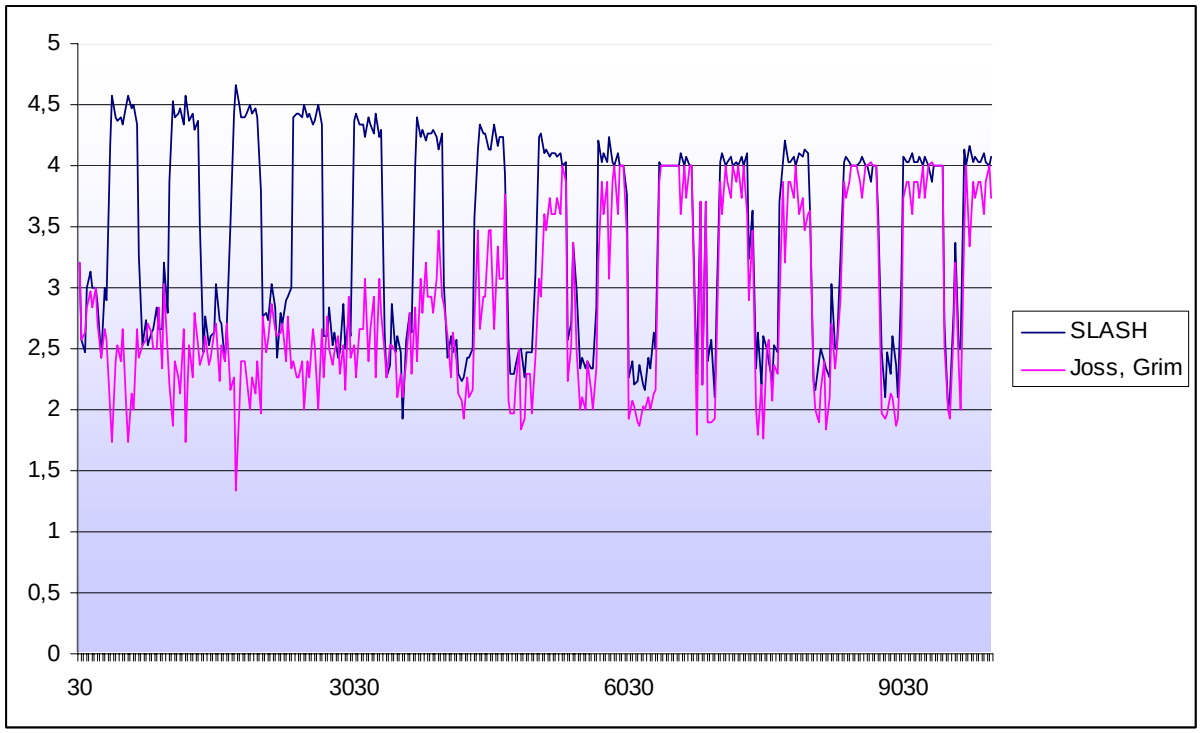

Figure 8.1.1

As we have already mentioned using these opponents makes for a very special case because of Grim. The only players that manage to play optimal against Grim are the players that are simple and can forgive defections. All other players will end up playing against a purely defective Grim opponent and may learn to play optimal in this case.

Game 3, Opponents: Tit for Tat, Defective

This third game is different from the two first in the case that it does not contain an opponent whose behaviour is difficult to learn. The optimal strategy against Tit for Tat is to cooperate while against the Defective player defection is the only rational choice.

Looking at the different players we see that Tit for Two Tat is the winner. Tit for Two Tat manages to play close to optimal against these opponents and handles the switch of opponents well. This is especially notable when switching from Defective to Tit for Tat. Let us first look at Tit for Tat to see how this switch is not handled optimally. Tit for Tat will

defect against the Defective player. When the opponent switches to Tit for Tat strategy, the player Tit for Tat will do a defection. The opponent will answer with a defection while the player will cooperate. This is somewhat the same situation as explained under game 2, where Joss defects against Tit for Tat. The result is that both the player and the opponent will defect and cooperate against each other as can be seen in Table 8.1.2.

Opponent	Tit for Tat				Defective				Tit for Tat			
Action Opponent	C	C	C	C	D	D	D	D	C	D	C	D
Action Player	C	C	C	C	C	D	D	D	D	C	D	C

Table 8.1.2

Tit for Two Tat handles this change better since it will forgive the first defection of the opponent Tit for Tat strategy. Table 8.1.3 shows this.

Opponent	Tit for Tat				Defective				Tit for Tat			
Action Opponent	C	C	C	C	D	D	D	D	C	D	C	C
Action Player	C	C	C	C	C	C	D	D	D	C	C	C

Table 8.1.3

Tit for Two Tat manages almost immediately to achieve mutual cooperation even after the opponent switches from Defective to Tit for Tat. This gives Tit for Two Tat an advantage. Of the other static players they all manage to get themselves in trouble with this combination of opponents. Joss will get the in the same situation as Tit for Tat, but even earlier. Joss will ruin the game for itself even more by doing its random defection against the Tit for Tat opponent. Grim will play optimal against the Defective player, but will never recover to play optimal against Tit for Tat. Grim will continue to defect when the opponent switches from Defective to Tit for Tat. The Tester player will eventually turn into a Tit for Tat strategy and will get the same disadvantages.

Of the stochastic learners, the Hierarchical player is doing well. In fact, using the subplayers Tit for Tat, Tit for Two Tat and Cooperative gives a lower score than both SLASH and I-SLASH. This instance of the Hierarchical player gets a score of 30 995 while SLASH gets 33 335 and I-SLASH gets 31 900. If the Hierarchical player managed to converge toward a single choice of strategy, the instance using Tit for Tat, Tit for Two Tat and Defective as subplayers would have been ideal. Against the Tit for Tat opponent, the Hierarchical player could in this case converge toward both Tit for Tat and Tit for Two Tat subplayer. However, converging can take a long time or not happen at all, since Tit for Tat and Tit for Two Tat looks practically identical from the point of view of the Hierarchical player. Against the Defective opponent, choosing defection would be the wisest choice. In this case all the subplayers will defect, and they will again look identical in the eyes of the Hierarchical

player. This results in the Hierarchical player not converging toward any of the subplayers, but give them all an identical chance of being chosen. In our tests, this ideal situation will never happen. Since it never converges, the Hierarchical player will not manage to avoid using the defective player against the Tit for Tat opponent. This results in an inefficient play against the Tit for Tat opponent, since it will make this opponent defect. If the Hierarchical player should switch to either Tit for Tat or Tit for Two Tat, we can expect one or more Sucker's payoffs as a result. The final result is a play where mutual defections will occur more often than strictly necessary. In the case using the subplayers Tit for Tat, Tit for Two Tat and Cooperative, this problem disappears against the Tit for Tat opponent. However, the Cooperative subplayer will be used against the purely Defective opponent, but the resulting punishments will be of such a significant nature that the probability of choosing this strategy will be lessened over time. Since they all share the properties of the Tit for Tat player, using the subplayers Tit for Tat, Joss and Tester will result in a bad score overall. Both SLASH and I-SLASH does well in this game. Studies of the behaviour of these players show that they manage to play optimal against both opponents and manage to handle the opponent switch well. I-SLASH, however, manages the change in a better manner because of its higher learning rate and ability to predict the opponent move.

Game 4, Opponent: SLASH

As already mentioned, using SLASH as an opponent results in a nonstationary environment. The special property of this environment is its ability to learn the strategies of the players and choose its actions thereafter. This makes a difficult environment to learn. According to Sandholm [8] learning against another learning entity creates difficulties since the learner does not have any a priori knowledge about the Iterated Prisoner's Dilemma such as a policy to encourage cooperation. In the case with SLASH, there is no optimal play. The best solution is to play in ways that result in mutual cooperation.

If we look at the fixed strategies first, we notice that some of them play bad against SLASH. This is especially notably with Tit for Two Tat. As already mentioned SLASH manages to use the properties of Tit for Two Tat for the good of its own. Tit for Two Tat, off course, gets punished for this with a resulting score of 34 090. This is the third worst of score this game. Reward Penalty is second worst, with a score of 35 531. Results around this player will be discussed later. The worst player of all is, again, Grim as a result of it total unforgivable nature. Tit for Tat, Joss and Tester will end up doing mutual cooperation with SLASH as opponent and will thus end up with a good score, respectively 21 795, 21 620 and 21 535.

The Hierarchical player show many interesting results in this setting. If the Hierarchy player uses the subplayers Tit for Tat, Tit for Two Tat and Cooperation, it will converge toward using Tit for Tat strategy, thus managing a mutual cooperation. The Hierarchy player will go away from using the Tit for Two Tat strategy because SLASH manages to use it in its favour. The Hierarchy player will also turn against using the purely cooperating player, since SLASH will learn to defect against this player. Figure 8.1.2 shows the probabilities of selecting each of the subplayers. The red line depicts the probability of selecting Tit for Tat, the green line depicts the probability of selecting Tit for Two Tat and the blue line depicts the probability of choosing Cooperative.

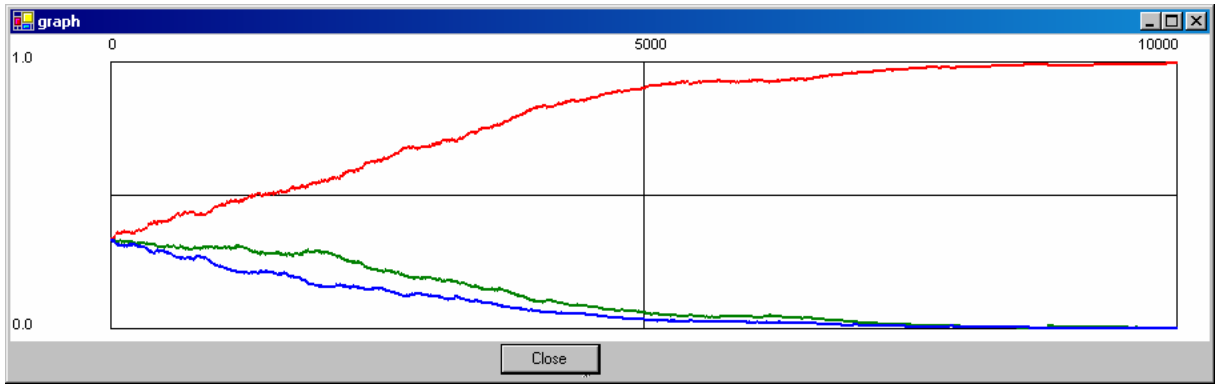

Figure 8.1.2

If we look at the case where the Hierarchical player uses Tit for Tat, Joss and Tester as subplayers, we will notice a totally different result. Here the Hierarchical player will not be able to see a distinction between its subplayers and will never converge toward using one of them. Figure 8.1.3 shows this. Red line is Tit for Tat, green line is Joss and blue line Tester.

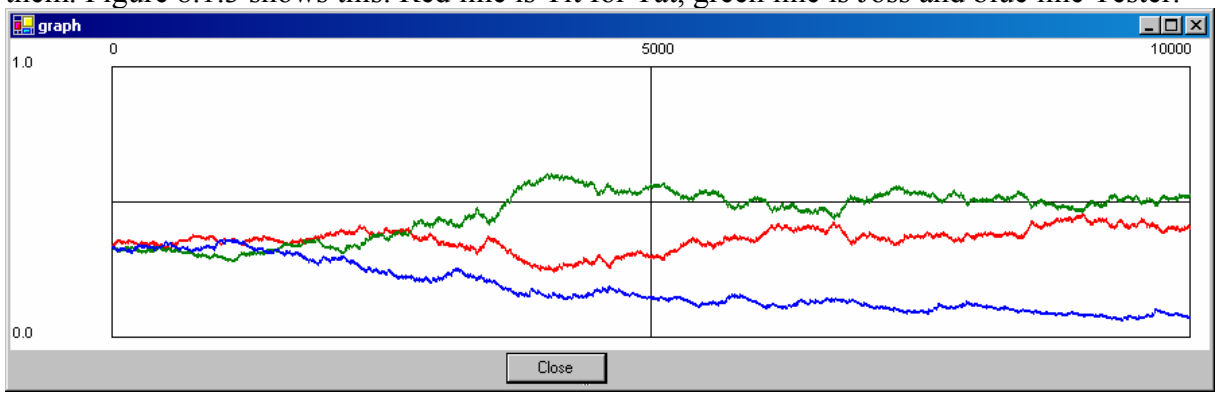

Figure 8.1.3

However, the end result will still be in the favour of the Hierarchical player; they will achieve mutual cooperation. The SLASH player will neither notice any significant differences in the player strategies and will choose to cooperate. Since mutual cooperation is achieved almost throughout the game, it will get a score almost equal to the score of Tit for Tat.

Playing SLASH against itself results in mutual cooperation. I-SLASH, however does not manage to play well against SLASH. I-SLASH tries to use SLASH in favour of itself. Test results show that I-SLASH always manages a better score than SLASH. In the long run, I-SLASH and SLASH will manage mutual cooperation, but this takes a lot of time, much longer than the 10 000 iterations shown here. One reason for this is that I-SLASH has lots of states that need to be updated, the same goes for SLASH.

In all the tests performed here we have used a Hierarchical player containing Tit for Tat, Tit for Two Tat, Reward Penalty player, Reward Inaction player and SLASH player. This hierarchy instance did overall bad, but several tests have shown that a Hierarchical player with a SLASH player as a subplayer will get a good result. SLASH will learn to play optimal against its opponents and will thus become an attractive choice for the Hierarchical player.

In these tests we have also used the Reward Penalty player as defined by Kehagias [7]. From the tests performed by Kehagias, the Reward Penalty player did not do well using the penalty values we have been operating with in this test. The fact is, this player will in our case not manage to converge toward mutual cooperation with itself. Converging toward mutual cooperation is one of the nice characteristics of this player.

Summary

If the opponent strategies are fixed, learning is much easier. Complications arise when the opponent is a learning entity itself. Fixed strategies like Tit for Tat or Tit for Two Tat is not effective in a nonstationary environment, unless they have built-in properties that give them advantages in the situation presented to them. An example of this is Tit for Two Tat against Joss and Grim. Grim and Tester are examples of fixed strategies that do not perform well in a nonstationary environment. Grim does not tolerate a single defection; if an opponent strategy should defect the overall game may suffer from it. Tester is a unique player in that it has its own set of strategy, but if a certain condition is met it switches to play exactly like Tit for Tat. This condition is when someone defects against it. If we put Tester in a nonstationary environment, it easily transforms itself into a Tit for Tat player, unless none of the opponent strategies defects. Tester therefore easily looses its uniqueness and becomes a Tit for Tat player with the advantages and disadvantages this involves.

The Hierarchical player manages well in a nonstationary environment, but in these tests it does not manage to converge toward a single choice of strategy against the different opponent strategies. Several reasons may be given for this. First of all, the hierarchical player does not remember its previous move(s). If the Hierarchical player moves toward one strategy against one opponent, playing against another strategy will maybe have the Hierarchical player move toward another strategy. It will not remember which strategy to use against different opponents. The more often environment switches occur, the more the learning will be forgotten and wasted. Adding history would make learning more efficient in these situations. Secondly, the Hierarchical player has a problem of differing too equal strategies. For instance is Tit for Tat and Tit for Two Tat behaving in equal manner unless the opponent should manage to misuse one of the strategies that the other avoids. If the opponent manages this, the Hierarchical player will converge toward the strategy that is not misused.

SLASH and I-SLASH remembers previous moves. This adds for a significant advantage both in a static and nonstationary environment. In a nonstationary environment SLASH does not have to unlearn previous choices of actions when a switch in the environment occurs. It simply adapts to the new environment. If the environment should switch back to the old strategy, SLASH will quickly realize, due to its history, that the current actions may be poor and choose to go back to its old choices of actions.

8.2 *Converging and behaviour*

We will here discuss the results presented in 7.2. We will present how different players converge against different opponent, and describe the effect of convergence. We will also discuss how well the players adapt after a switch in the environment.

Table 7.2.1 shows how well each player is able to perform, measured by their average penalty. This table shows how well FSA players are playing compared to SLASH/I-SLASH and also the Hierarchical player is presented with one combination of subplayers. The opponents are both single FSA and switching strategies to create a nonstationary environment. It is likely that some of the FSA are made to correspond to some of the other FSA strategies. Tester examines whether the opponent is exploitable as is the case with Tit for Two Tat, and will continue to alternate between defection and cooperation if it is possible, or else it will play like Tit for Tat. Grim plays like defective player, but starts with cooperation and continues until someone defects, if it should meet players like Tit for Tat.

As we can see in Table 7.2.1, I-SLASH and SLASH are not able to play optimal against any players. An important reason for this is because SLASH/I-SLASH will not converge entirely. The players have a percentage no more than 99% to choose the optimal action. While playing against opponents like Tit for Tat, Tit for Two Tat or Cooperative, the optimal play will make the opponents always cooperate, while the player plays different strategies on each of the opponents. Discovering the switch of opponents requires a different move once in a while. Even though I-SLASH did not play optimal against any of the players solely, it achieved distinctively the best result against them in a switching environment.

I-SLASH consists of different states, where each state represents former moves. In many cases such a state may represent a certain strategy. If the opponents are Tit for Tat and Defective, a state where the opponent cooperated no matter what can only occur if the opponent is Tit for Tat, and a state where the opponent defected even after the player cooperated in the former round, can only occur with Defective as the opponent. Since all states have their own LA, they are able to learn the perfect play against these opponents. However, there is an exception; this is when the players and opponents behave identically. Suppose the history consist of only defection. Then it is impossible to verify whether you are playing against Tit for Tat or Defective. If the opponent is Tit for Tat, the player is playing an inferior strategy, and while playing against Defective, this is the optimal strategy. In this case the probability of choosing one action may be updated in both towards cooperation as well as towards defection, and the state needs time to adjust after each switch. This is the main reason why I-SLASH are doing better against the opponents Tit for Tat and Defective, than against the opponents Tit for Tat and Tit for Two Tat or the opponents Tit for Tat, Tit for Two Tat and Cooperative. Even though I-SLASH looses more in these switches, no other player are able to approach the results of I-SLASH. Obviously other players are not able to identify the switches at all.

I-SLASH gets the best average result in these tests. These tests are performed with 200 000 and 240 000 rounds, which is a very long time and are not comparable with the actual results

of Axelrod's tournament. I-SLASH will play better than pure chance after just a few dozen rounds, but will need thousands of rounds to reach its best play, depending on the opponents.

Figure 7.2.1 shows how I-SLASH improves while playing against the opponents Tit for Tat and Defective. I-SLASH improves after every switch, especially against Defective, which is easier to identify. Figure 7.2.2 shows two switches. These switches occur after I-SLASH already has learned each of the strategies. The game has already played 160 000 rounds and we compare the results with the values I-SLASH have achieved against each of the opponents alone, using the values presented in Table 7.2.1. The results show that I-SLASH gets an increase in penalty of 280 on these two switches, and most of this is on the switch from Defective to Tit for Tat. The reason for this is because I-SLASH is based on states of history, where every history that includes a cooperation of the opponent will under no circumstances be other than Tit for Tat, but if I-SLASH should experience mutual defection with Tit for Tat, it will get into a state where it usually are playing against Defective, and where it usually continues with mutual defection. This mutual defection will give a penalty of 4 instead of 2. A mistake while playing against Defective will give a penalty of 5 instead of 4. This increase in penalty is just the half compared with playing against Tit for Tat. An increase of 280, due to the switch is a small penalty, and shows that I-SLASH is able to adapt quickly after a switch.

A fixed structure automaton will gain little or no penalty on a switch in the environment, but will in many cases not be able to play optimal against the opponent. Let us take Tit for Tat as an example. Tit for Tat plays optimal against both itself and Defective individually. Tit for Tat versus itself will always result in mutual cooperation. Tit for Tat against Defective will always result in mutual defection. However, if Tit for Tat and Defective are put together as a nonstationary environment, matters are different. The reason comes clear if we look at an opponent switch from Defective to Tit for Tat. Tit for Tat will defect the first round against the opponent Tit for Tat, and from then on, alternate between cooperation and defection, which is not optimal. If the opponent Tit for Tat defects on the last round before Defective plays its rounds, then the rest of the game will result in mutual defection. Tit for Two Tat handles this problem better than Tit for Tat due to its forgiving nature. Tit for Two Tat, however are playing inferior to Tit for Tat against Tester and Random. It will also easily be exploited by learning algorithms like SLASH/I-SLASH.

The Hierarchical player may include optimal FSA which do not need to be learned. But the top-player will always need time to adjust after a change. The Hierarchical player may converge. If the Hierarchical player converges towards cooperation against Tit for Tat, it will not notice if the environment has switched to a purely cooperative strategy. The Hierarchical player also needs longer time to learn, because it adjusts its probabilities only after a series of rounds, usually 5.

Figure 7.2.3 shows the Hierarchical player against Random, Joss and Defective. Each of these opponents has one corresponding subplayer in the hierarchy that is able to play optimal against it. Even though I-SLASH does not play optimal against any of the opponents, it still scores better overall compared to the Hierarchical player. The Hierarchy player plays optimal against Defective, because both of its subplayers will do that, but looses too much

against the other players, and in the switch. The subplayers in the hierarchy are already trained to play an optimal game against its corresponding opponent, but this experiment shows that loss in all the switches are more than the double compared to I-SLASH.

Summary
In a static environment, the optimal play is usually carried out by a simple FSA strategy. Some simple FSA strategies are even able to play optimal against a few instances of a nonstationary environment. I-SLASH will never converge and is therefore not able to play optimal, but are playing overall good against all the opponents. I-SLASH will need time to learn, and after enough rounds, I-SLASH is able to handle a switch of the opponents' strategy with only a small increase of penalty, and far better than the Hierarchical player. The Hierarchical player may converge, and needs more time to adapt after a switch in the environment has occurred. The effectiveness of the hierarchical player is also more dependent of its subplayers and the opponents compared to I-SLASH which is overall persistently good.

8.3 *Further work*

There are many interesting aspects to study related to learning automata, and we will probably see many new automata appear in the years ahead. Several issues in this thesis could be subject to further work. We will briefly introduce a few aspects in this section.

- Learning automata have a wide are of usage. Experiments have been done among others in automated highway systems, intelligent vehicle control [12], network control and telephone traffic routing. An interesting study could be to adjust and implement SLASH, I-SLASH and the hierarchy player in another environment than the IPD. It might be possible to adjust and implement it in some of the areas mentioned above.

- There exist many different update algorithms for probabilistic learning automata. We have experienced with Reward Penalty, Reward Inaction and Inaction Penalty. It could be interesting to experiment with other learning algorithms within the framework of SLASH / I-SLASH and the hierarchical player as well.

- We have already implemented SLASH as a top-player in I-SLASH. Here it is proven that SLASH handles this role well. Another interesting experiment could be to implement a SLASH player as a top-player of the hierarchy player.

- I-SLASH could be developed to support more levels in the hierarchy. This would make the player more advanced, but it will also need more rounds to learn the strategies, since there will be more states to update.

- New algorithms and techniques will always be valuable to improve the learning time, and the accuracy of an automaton.

9 Conclusion

In this thesis we have developed and evaluated three new algorithms of learning automata. These algorithms are developed to suite a dynamic switching environment. The environment used is the Iterated Prisoner's Dilemma where the opponent may switch strategy while playing. Until now, well known existing stochastic algorithms have not been fast enough to adapt after a switch has occurred. Fixed structure automata are usually able to switch faster but are not able to learn the optimal strategy of all the opponents.

The first algorithm we developed, the Hierarchical player, is able to adapt to a certain point, but are very dependent of which subplayers it consist of. After a series of tests, it shows to do fairly well, but do not score better than all the opponents. The strategy based on states of history, SLASH, is able to score better than any prior strategies on the average tests. SLASH is able to learn advanced strategies that it will use against its opponents. This is a property not shared by any other non-history automata we have used. SLASH also adapts to new strategies quickly. The improved version of SLASH, called I-SLASH, scores even better than its predecessor and learn faster.

The Hierarchy player is developed with the ability to converge entirely towards a single strategy. In some cases this lead to a convergence without being able to adapt when the opponent switches its strategy. SLASH and I-SLASH are developed not to converge entirely in order to avoid this situation, and will therefore not play optimally. But it is often able to score close to optimal.

In the hypothesis H1 we stated that interconnected learning automata will play better in a nonstationary environment than existing automata. We defined a prediction P1-1 that would be used to evaluate the validity of the hypothesis H1. This prediction states that interconnected learning automata will get an average better score than existing automata. If we look at the Hierarchical player the prediction is true in many cases. Since the score is dependent on the choice of strategies the player has, the score will not always be optimal. As for the SLASH player the prediction is true since it gets an average score that is better than any other strategies. SLASH beats the winner in the tournament put up by Axelrod, Tit for Tat. The prediction P1-1 is true in many of our tests and our hypothesis H1 is strengthened in validity.

We also defined a hypothesis H2 which stated that game history will improve the effectiveness of an interconnected automaton. With this hypothesis we defined two predictions to validate the validity of H2. The first prediction P2-1 states that interconnected automata with history will get a better score than other interconnected automata. In our case other interconnected automata without history is the instances of the Hierarchical player. We have performed tests where several instances of the Hierarchical player and SLASH have played against different opponents. In this case SLASH manages to achieve a better score than the Hierarchical player. SLASH also manages to play well against the Hierarchical player as opponent. We can therefore state that P2-1 is valid in these tests. The second prediction, P2-2, states that interconnected automata with history will get a better score than

other strategies. SLASH manages to play close to optimal against any strategies we have tested. For instance it learns to play close to optimal against Tit for Two Tat by alternating between cooperation and defection. However, it does not manage to play ideal against Grim because of the learning nature SLASH consists of. Therefore it may be beaten in some cases by non-learning algorithms. Overall though, the SLASH player gets a better score and we can safely state that P2-2 is valid. With both P2-1 and P2-2 valid, the hypothesis H2 is strengthened in validity.

Learning automata already have a wide area of usage, and the use of artificial intelligence is expanding. In many cases where automata operate, the environment is not static; rather it is dynamic and unpredictable. There will be a considerable need for an accurate decision-maker with adaptive powers. I-SLASH is our best contribution to this environment. With adjustments to fit other environments as well, it could have a great utilitarian value to existing systems as well as to new technology.

10 Abbreviations

Coop = Cooperative player
Def = Defective player
FSA = Fixed Structure Automata
ILA = Interconnected Learning Automata
IP = Linear Inaction Penalty
IPD = Iterated Prioner's Dilemma
I-SLASH = Interconnected Stochastic Learning Automata with States of History
LA = Learning Automata
PD = Prisoner's Dilemma
RI = Linear Reward Inaction
RP = Linear Reward Penalty
SLASH = Stochastic Learning Automata with States of History
TFT = Tit for Tat
TFTT = Tit for Two Tat

11 References

[1] Greenwald, Amy. *Modern Game Theory: Deduction vs. Induction*, January 1997

[2] Axelrod, Robert and Hamilton, William D. *The Evolution of Cooperation,* March 1981

[3] Heap, Shaun P. H. and Varoufakis Yanis. *Game Theory: A Critical Introduction* p. 85, Routledge 1995. ISBN 0-415-09402-X

[3] Heap, Shaun P. H. and Varoufakis Yanis. *Game Theory: A Critical Introduction* p. 169, Routledge 1995. ISBN 0-415-09402-X

[4] Tsetlin, M.L., *Automaton Theory and Modeling of Biological Systems*. New York: Academic Press, 1973

[5] Ünsal, Cem, *Intelligent Navigation of Autonomous in an Automated Highway System: Learning Methods and Interacting Vehicles approach*. 1997

[6] Narendra K., Thathachar M.A.L., *Learning Autonama – An Introduction*, Prentice-Hall, 1989

[7] Kehagias, Athanasios. *Probabilistic Learning Automata and the Prsioner's Dilemma*, 1994

[8] Sandholm, Thomas. W and Crites, Robert H. *Multigagent Reinforcement Learning in the Iterated Prisoner's Dilemma*

[9] Woolridge, Micheal J. *An introduction to multiagent systems,* Wiley 2002. ISBN 0-471-49691-X

[10] *McGraw-Hill Encyclopedia of Science & Technology 6th Edition*, McGraw-Hill Book Company. ISBN 0-07-079292-5

[11] Thathachar M.A.L., Sastry P.S. *Networks of Learning Automata. Techniques for Online Stochastic Optimization,* Kluwer Academic Publishers, 2004.

[12] Üncal, Cem. *Intelligent Navigation of Autonomous Vehicles in an Automated Highway System: Learning Methods and Interacting Vehicles Approach,* January 1997