Master in Artificial Intelligence (UPC-URV-UB)

Master of Science Thesis

Learning Probabilistic Finite State Automata For Opponent Modelling

Toni Cebrián Chuliá

Advisor/s: René Alquézar and Alberto Sanfeliu

January 17th, 2011

A Sandra

Contents

1 Introduction **5**
- 1.1 Motivation . . . 5
- 1.2 Problem definition and goals . . . 6
- 1.3 Thesis overview . . . 7

2 Problem Description **9**
- 2.1 The Game Environment . . . 9
- 2.2 Interacting agents . . . 10
- 2.3 Repeated games . . . 11
  - 2.3.1 Games. Normal Form . . . 11
- 2.4 Opponent Modelling . . . 13
  - 2.4.1 Learning a model . . . 13
  - 2.4.2 Model assumption . . . 14
- 2.5 Learning the opponent's transducer . . . 14
  - 2.5.1 ALERGIA . . . 15
  - 2.5.2 MDI . . . 15
  - 2.5.3 GIATI algorithm . . . 15
- 2.6 Defeating the opponent . . . 16
  - 2.6.1 Utility functions . . . 16
- 2.7 Example . . . 17

3 Learning Probabilistic Finite State Automata **21**
- 3.1 Introduction . . . 21
- 3.2 Symbols, Languages and Grammars . . . 21
- 3.3 Finite State Automaton . . . 22
  - 3.3.1 Quotient Automaton . . . 22
  - 3.3.2 Probabilistic Automata . . . 22
- 3.4 Algorithms . . . 23
  - 3.4.1 Identifying regular languages . . . 23
  - 3.4.2 RPNI . . . 25
  - 3.4.3 ALERGIA . . . 27
  - 3.4.4 MDI . . . 28

4 Opponent Modelling **31**
4.1 Defeating an Opponent 31
4.2 The GIATI algorithm 31
4.2.1 Some terminology 31
4.2.2 Inferring Finite-State Transducers 34
4.2.3 Applying GIATI in the repeated games scenario 35
4.3 Markov Decision Processes 35
4.3.1 Links between Probabilistic Mealy Machines and MDP . 37
4.3.2 Infinite horizon models 38
4.3.3 Finding optimal policies 39
4.4 Summary 40

5 Experiments **43**
5.1 The experiments 43
5.2 Prisoner's Dilemma 43
5.2.1 Round length 44
5.2.2 Match length 46
5.2.3 Alpha 46
5.3 RoShamBo 48
5.3.1 The experiment 48

6 Conclusions **53**
6.1 Future work 54

A JSON format for a game **57**

B Automata for the RoShamBo experiments **59**

Chapter 1

Introduction

1.1 Motivation

Artificial Intelligence (AI) is the branch of the Computer Science field that tries to imbue intelligent behaviour in software systems. In the early years of the field, those systems were limited to big computing units where researchers built expert systems that exhibited some kind of intelligence. But with the advent of different kinds of networks, which the more prominent of those is the Internet, the field became interested in Distributed Artificial Intelligence (DAI) as the normal move.

The field thus moved from monolithic software architectures for its AI systems to architectures where several pieces of software were trying to solve a problem or had interests on their own. Those pieces of software were called Agents and the architectures that allowed the interoperation of multiple agents were called *Multi-Agent Systems* (MAS). The agents act as a metaphor that tries to describe those software systems that are embodied in a given environment and that behave or react intelligently to events in the environment.

The AI mainstream was initially interested in systems that could be taught to behave depending on the inputs perceived. However this rapidly showed ineffective because the human or the expert acted as the knowledge bottleneck for distilling useful and efficient rules. This was in best cases, in worst cases the task of enumerating the rules was difficult or plainly not affordable. This sparked the interest of another subfield, Machine Learning and its counter part in a MAS, Distributed Machine Learning. If you can not code all the scenario combinations, code within the agent the rules that allows it to learn from the environment and the actions performed.

With this framework in mind, applications are endless. Agents can be used to trade bonds or other financial derivatives without human intervention, or they can be embedded in a robotics hardware and learn unseen map configuration in distant locations like distant planets. Agents are not restricted to interactions with humans or the environment, they can also interact with other agents themselves. For instance, agents can negotiate the quality of service of

a channel before establishing a communication or they can share information about the environment in a cooperative setting like robot soccer players.

But there are some shortcomings that emerge in a MAS architecture. The one related to this thesis is that partitioning the task at hand into agents usually entails that agents have less memory or computing power. It is not economically feasible to replicate the big computing unit on each separate agent in our system. Thus we can say that we should think about our agents as *computationally bounded* , that is, they have a limited amount of computing power to learn from the environment. This has serious implications on the algorithms that are commonly used for learning in these settings.

The classical approach for learning in MAS system is to use some variation of a *Reinforcement Learning* (RL) algorithm [BT96, SB98]. The main idea around those algorithms is that the agent has to maintain a *table* with the perceived value of each action/state pair and through multiple iterations obtain a set of decision rules that allows to take the best action for a given environment. This approach has several flaws when the current action depends on a single observation seen in the past (for instance, a warning sign that a robot perceives). Several techniques has been proposed to alleviate those shortcomings. For instance to avoid the combinatorial explosion of states and actions, instead of storing a table with the value of the pairs an approximating function like a neural network can be used instead. And for events in the past, we can extend the state definition of the environment creating dummy states that correspond to the N-tuple $(state_N, state_{N-1}, \ldots, state_{N-t})$

1.2 Problem definition and goals

Given the problems before mentioned this thesis studies the effect of using a model based approach to learn in a competitive agent environment. In order to model the environment, in this case, an opposing agent, probabilistic finite state automata will be employed. This will limit the range of possible hypothesis when searching the state space. For illustrative purpose, the environment where our agents will live is an abstraction of a competitive setting modelled as a game theoretic model.

In this thesis we are going to explore the following related issues:

1. Some state-of-the-art algorithms for learning probabilistic automata will be presented and discussed. Comparison of the learning abilities of each algorithm will be under study

2. Agents can be seen as highly complicated transducers. That is, they are software artifacts that after receiving some sensory inputs react to the environment performing some actions. How can the implicit transducer that an agent hides within its machinery be inferred, will be researched.

3. Finally, when a working hypothesis is obtained, this model must be exploited in order to take an advantage in this setup. But our agent has to keep in mind that this hypothesis might not be the underlying transducer and that some inexactitudes may happen, both in the structure of the model or in the current state. How different models of complexity degrade over time will be studied because we are mainly interested on winning the agent competition

This Master's Thesis tries to explore beyond the point where the thesis of David Carmel ended. In his thesis he tries to infer deterministic automata restricted to an alphabet with only 2 symbols. In this work, the general framework is extended by allowing arbitrary alphabet length and by trying to infer *stochastic deterministic transducers*. We have not been able to find online the before mentioned thesis but the main lines of research can be learned by reading the references [CM96b, CM96a, CM98, CM99].

1.3 Thesis overview

This thesis is organized as follows:

- **Chapter 2** outlines the different areas this thesis relies on. A very simple example of the steps and computations is used for illustrative purposes.

- **Chapter 3** introduces the necessary automata concepts to understand the building blocks of the methodology and exposes the algorithms used to learn the PDFA.

- **Chapter 4** presents the algorithmic theory that allows to infer transducers using automata. Once we have our working hypothesis of the model our opponent is using for choosing movements, we should exploit this knowledge. To that end we show how can transducers be translated into and Markov Decision Processes.

- In **Chapter 5** we study how our setup is able to infer opponent strategies and the rate and effectiveness of the learning process.

- **Chapter 6** presents the conclusions derived from this study

Chapter 2

Problem Description

2.1 The Game Environment

In order to test our ideas we need an environment where two competing agents interact with each other. Figure 2.1 illustrate all the elements that are involved in the experimental setup.

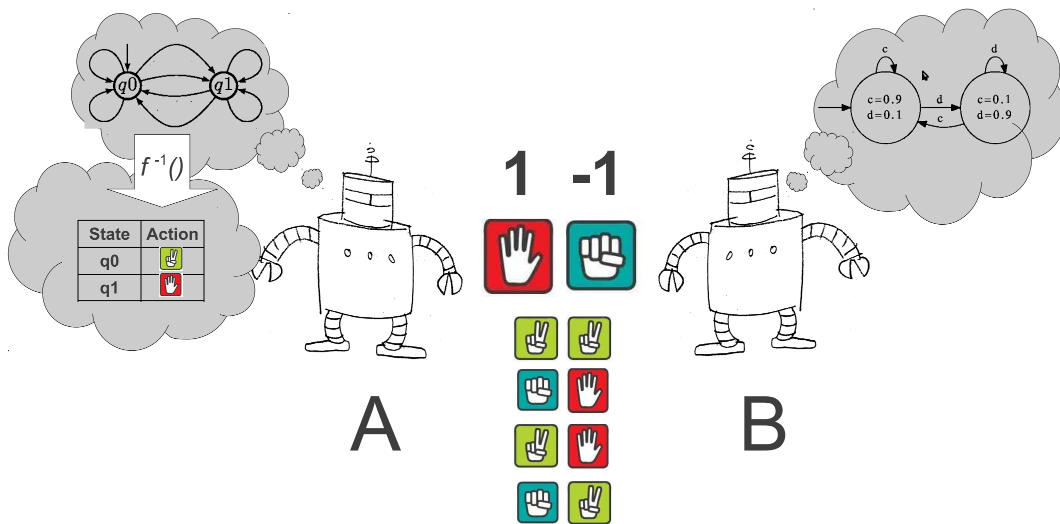

Figure 2.1: Two opposing agents, one that learns and the other that has a fixed strategy are competing playing the RoShamBo game

Let's review the main components:

- Agents are playing a repeated game in the sense that is defined by the Game Theory field. Players play simultaneous games emitting symbols, and after seeing the pair of symbols, an external referee assigns payoffs to each player. In the figure, agents are playing RoShamBo and the symbols they can emit are either Paper, Rock or Scissors. The current movement

gives one point to our learning agent and -1 points to our fixed agent because in RoShamBo, Paper wins Rock.

- There are two **opposing agents**. In the figure they are represented by the letters A and B. The B-agent is the agent we are trying to model. It has a probabilistic Moore Machine as its inner core strategy shown in the upper right cloud. That strategy is fixed and will not change during the game and is governed by the symbols that the A-agent sends.
- The A-agent is our learning agent. After some iterations with the B-agent it creates a model hypothesis depicted in the upper-left cloud. This automaton translates nicely to a transducer which this agent believes is the strategy of the opposing agent. Once the agent has this hypothesis it devises a counter strategy that is used to play the game until the end

The following sections address each element with more detail.

2.2 Interacting agents

One of the most interesting areas of the Artificial Intelligence field is Multi Agent Systems (MAS). According to [Woo02],

MAS Multiagent systems are systems composed of multiple interacting computing elements, known as agents.Agents are computer systems with two important capabilities. First, they are at least to some extent capable of autonomous action of deciding for themselves what they need to do in order to satisfy their design objectives. Second, they are capable of interacting with other agents not simply by exchanging data, but by engaging in analogues of the kind of social activity that we all engage in every day of our lives: cooperation, coordination, negotiation, and the like.

Another definition about what an agent is can be found in [RN03]

> An **agent** is anything that can be viewed as **perceiving** its environment through **sensors** and acting upon that environment through **actuators**

A *rational agent* is an agent that does the right thing according to a *performance measure* that evaluates a given sequence of environment states. Note that the performance measure evaluates environment states, not agent states. We can thus define rationality as dependant on four things:

1. The performance measure that defines the criterion of success
2. The agent's prior knowledge of the environment

3. The actions that the agent can perform

4. The agent's percept sequence to date

This again leads to the definition of a *rational agent*.

> For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has [RN03]

The built-in knowledge is usually called the agent program. This program guides how the agent picks actions based on the environment. Depending on the procedures used by the agent, agent programs can be classified in four basic kinds of agent programs:

1. Simple reactive agents

2. Model-based reactive agents

3. Goal-based agents

4. Utility-based agents

In this thesis we are going to explore *Model-based reactive agents*. For a more exhaustive explanation of the different kinds of agent programs check [PW04].

When an agent can be ascribed to the model-based reactive paradigm we expect that the agent should maintain some sort of *internal state* that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current environment state. In our case, the model of the world will be an inferred transducer and the state our agent should maintain is in which state our transducer currently is.

2.3 Repeated games

2.3.1 Games. Normal Form

A game [LR57] is a mathematical structure that comprises three elements:

1. A set of rules with the inner working of the game

2. A set of available moves to each player depending on the state of the game

3. A function with the outcomes depending on the state of the game

A game is usually used to represent a conflict between several contending players. This mathematical formulation allows the researcher to employ mathematics to study the outcome of several policies or strategies. In this thesis, moves from each player do not depend on the game state, all movements are available in whichever instant of the game. Moreover, outcomes or *payoffs* are also readily available when both players perform a move. These constraints allows us to utilize in this thesis the *normal form* description of a game. When using the *normal form*, games are described by using a matrix where the columns show the available movements for one player and the rows the movements for the other player. Cell intersections are used to represent the *payoffs* of each player. When the payoff is different for each player, a tuple is used instead, with the row player payoff as its first element and the column player payoff as its second element.

The competitive interaction between agents can thus be formalized as a repetition of a simple *two-player game*, where on each movement agents interact with each other by simultaneously performing an action. The complete set of pairs of actions can be aggregated to denote the history game. Let's see some examples of games.

Prisoner's dilemma

Prisoner's dilemma is a two-player game, where each player has two possible actions $A = \{c, d\}$ and the payoff for each player, p_1, p_2 is described in the payoff matrix 2.1. For the Pirsoner's Dilemma the game theoretical strategy

	c	d
c	(3,3)	(0,5)
d	(5,0)	(1,1)

Table 2.1: The payoff matrix for the Prisoner's dilemma game

is to defect always. This is the optimal strategy for games that only involve one interaction, but for repeated games, it is of the interest of the agents to cooperate in the long run because this strategy rewards the agents the most.

RoShamBo

RoShamBo is a popular game that is seemingly used across cultures to dilucidate minor conflicts or choices. Each player can perform one of three actions $A = \{paper, stone, scissors\}$ where each action beats or is beaten by the other two actions. This can be explicitly shown in the payoff matrix 2.2.

RoShamBo has no stable strategy neither for the single interaction nor for the repeated game interactions. It is also a zero sum game where what one player wins, the other player loses.

	paper	stone	scissor
paper	(0,0)	(1,-1)	(-1,1)
stone	(-1,1)	(0,0)	(1,-1)
scissor	(1,-1)	(-1,1)	(0,0)

Table 2.2: The payoff matrix for the RoShamBo game

2.4 Opponent Modelling

This thesis is mainly about learning the underlying strategy of an opposing opponent. In game theoretic terms it is called *opponent modelling*. We want to infer the most accurate model possible of our opponent because once we have that picture we can create explicit strategies specially tailored to counter act this opponent's strategies.

2.4.1 Learning a model

Learning a model of an opponent is a kind of unsupervised learning environment. We do not have a set of labeled pairs of sensory input and opponent state. Instead what we have is the sensory input, that is the set of actions exchanged, and the game outcomes that are somehow related to the agent's state.

In opponent modelling we are trying to infer an abstracted description of the player or the player's behaviour during the game. Opponent modelling is widely used in domains where the competitive nature of the agents are central like in Poker. Opponent modelling techniques are employed there to classify an opponent in an abstract sense as *aggressive*, *rock* or *calling station* [1]. Alternatively, opponent modelling can go a step further in granularity and try to predict whether a given player will *call, check* or *raise* in a Poker round.

For learning the opponent's model, you usually have to employ some kind of *Reinforcement Learning* algorithm (RL)[SB98]. RL algorithms are algorithms that learn which actions our agent has to perform in a given environment to accomplish a given goal usually modelled as a numerical reward. This family of algorithms, amongst we can find *TD-learning, Q-learning and SARSA* [BT96], try to infer a function that goes from the state of environment to the actions that should be performed. This function can be as simple as a table with pairs state/action or in cases where the combinatorial explosion of states and actions make infeasible to store such tables, this function can be approximated by an auxiliary function flexible enough to model those subtleties. Those approximating functions are usually created by training neural networks [Hay08].

[1] For a more comprehensive poker player descriptions check [Skl99]

2.4.2 Model assumption

In this thesis, our main assumption is that our opposing agent has an algorithmically encoded strategy to play the game. This algorithm belongs to the family of functions that can be described as Stochastic Finite State Transducer (SFST). Basically the model we are assuming is that our opponent reacts to our actions deterministically changing to another state in its automaton. In order for the agent to generate its next action, each state has a discrete probabilistic distribution where symbols will be generated according to that distribution. It is also known as a *Stochastic Moore Machine*.

Moore machines can be directly translated into Mealy machines and the other way around. For our learning purposes it is more convenient that this Stochastic Moore Machine be described as a Stochastic Mealy Machine since this translation allows us to use all the already known machinery to deal with Markov Decision Processes. Transforming a Moore machine into its Mealy counterpart can be easily accomplished by means of appending the symbol emitted in the state to each of the incoming transitions. Since the Moore machine is Stochastic, this propagation carries the probability associated and generates in turn a Stochastic Mealy Machine.

Let's see a concrete example. In Figure 2.2 we can see an example of such Moore machines representing an agent that "almost" always emits the same symbol that triggered the transition. After performing the output symbol

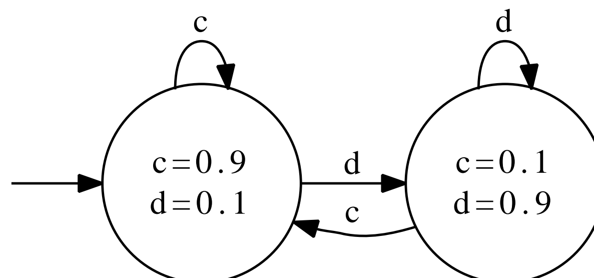

Figure 2.2: Stochastic Moore Machine

propagation on each step, we obtain the Stochastic Mealy machine that can be seen in Figure 2.3.

Learning this kind of transducers that represent the computational strategy of the opposing agent will be the objective of the learning agent.

2.5 Learning the opponent's transducer

In order to learn the opponent transducer, rounds of the game will be played and the results of each round will conform the training data for our PDFA learning algorithms. After a PDFA has been obtained and using the GIATI

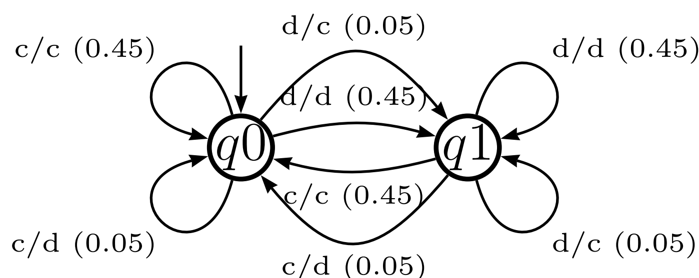

Figure 2.3: Transformed Mealy machine from the original Moore machine

algorithm a transducer is also obtained. This transducer is similar to a Markov Decision Process for which known algorithms for proper control exist.

This thesis is concerned with the comparison of different PDFA learning algorithms and their effect on the competitive environment of the proposed games. Two main algorithms are going to be studied, ALERGIA and MDI.

2.5.1 ALERGIA

The ALERGIA algorithm by Carrasco and Oncina [CO94] is an extension of the non-stochastic algorithm RPNI presented in [Onc92].

2.5.2 MDI

The MDI algorithm ([Tho00]) is conceptually similar to the ALERGIA algorithm but improves results by taking into account global features whereas ALERGIA is concerned in local ones.

2.5.3 GIATI algorithm

After obtaining an automaton, we use the GIATI algorithm for obtaining the transducer. In order to learn the **Stochastic Finite State Transducer** SFST we employ the GIATI algorithm described in [CVP05]. Given an input alphabet Σ and an output alphabet Δ our training corpus consist of words formed by (s,t) $\in \Sigma^+ \times \Delta^+$

1. Each training pair (s,t) is transformed into a string **z** from an extended alphabet Γ (strings of Γ-symbols) yielding a sample S of strings $Z \in \Gamma^*$

2. A stochastic regular grammar G is inferred from S

3. The Γ-symbols of the grammar rules are transformed back into pairs of source/target symbols/strings (from $\Sigma^* \times \Delta^*$).

The main problem using the GIATI algorithm is that the mapping from the training pairs to the new alphabet, the labeling function $L : \Sigma^* \times \Delta^* \to \Gamma^*$

and the inverse function are not specified and they must make sense in the problem at hand. In our problem, we don't have such nuances since, in the games we are studying, both agents perform only one action on each turn and thus, no problem of alignment between strings has to be inferred.

In Figure 2.4 we can see the machine presented in Section 2.4.2 transformed into its equivalent Probabilistic Deterministic Finite State Machine over the new vocabulary $\Gamma : \{cc, cd, dc, dd\} \equiv \{x, y, w, z\}$.

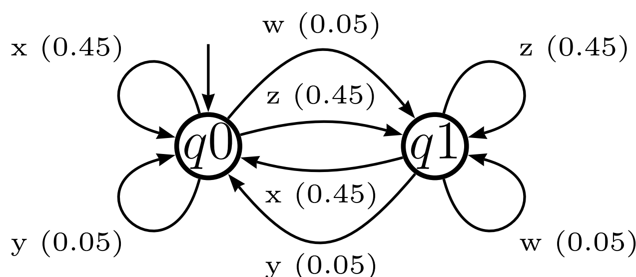

Figure 2.4: The Mealy Machine has been transformed into a Probabilistic DFA with an extended alphabet

2.6 Defeating the opponent

Once we have an opponent's model hypothesis, our task is to generate the corresponding actions that helps us to govern the automaton. This is easily accomplished given the fact that a Mealy Machine, a transducer, can be easily converted to an equivalent Markov Decision Process. So our first step will be to perform that conversion and after that use one of the algorithms for inferring the best policy for the MDP.

2.6.1 Utility functions

In order to solve the *best response* problem we have to define the utility function that our agent is going to use to evaluate a result. This utility function is related to each of the payoffs on each game stage and to the remaining horizon in the whole game.

There are several possible utility functions that drive the search of actions in the MDP, here are a couple for illustrative purposes.

- The **discounted-sum** function:

$$U_i^{ds}(s_1, s_2) = (1 - \gamma_i) \sum_{t=0}^{\infty} \gamma_i^t u_i(s_1(g_{(s_1,s_2)}(t)), s_2(g_{(s_1,s_2)}(t)))$$

  Where U_i is the utility function for player i when hi is playing according to strategy s_1 and his opponent is using strategy s_2.

- The **limit-of-the-means** functions:

$$U_i^{lm}(s_1, s_2) = \lim_{k\to\infty} inf \frac{1}{k} \sum_{t=0}^{k} u_i(s_1(g_{(s_1,s_2)}(t)), s_2(g_{(s_1,s_2)}(t)))$$

  Where again, U_i is the utility function for player i when he is playing according to strategy s_1 and his opponent is using strategy s_2.

2.7 Example

To illustrate all the process involved, let's develop an example. Each step will be explained in further detail in next sections, this is a high level summary of the info that has to come.

Let's start assuming that our agents are competing in a Prisoner's Dilemma competition. The agent that has the fixed strategy is playing the probabilistic Tit-For-Tat strategy. That means that this agent, almost always, repeats your last move. This agent has been seen before in Figure 2.2 and we reproduce it here for easiness in following the explanation (Figure 2.5).

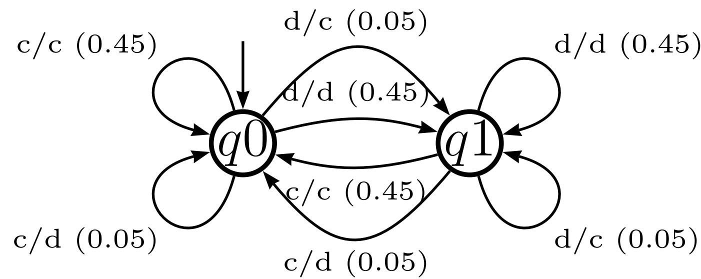

Figure 2.5: Probabilistic transducer (Tit-for-Tat)

Since this is our first iteration, our learning agent has no information about what is the structure of its opponent, so the best approach is to generate movements at random. The initial structure can be seen in Figure 2.6. Keep in

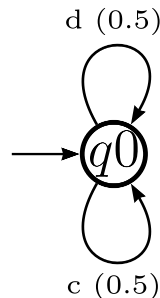

Figure 2.6: Random Automata

mind, that our random automaton is only a concise way to represent a uniform

distribution of symbols. For our learning agent the model used to generate movements can be whatever structure, automata, functions or a translation table.

Let's advance some terminology. Our objective, is that our learning agent outperforms its opponent in a *Game*. A game is composed of several *rounds*. Each round has a fixed number of *matches* with a fixed number of movements for each match (Figure 2.7) where each agent picks a symbol, and a reward is given as an answer of this action. For our example, the game will consist on two rounds with 25 moves each distributed on 5 matches per round.

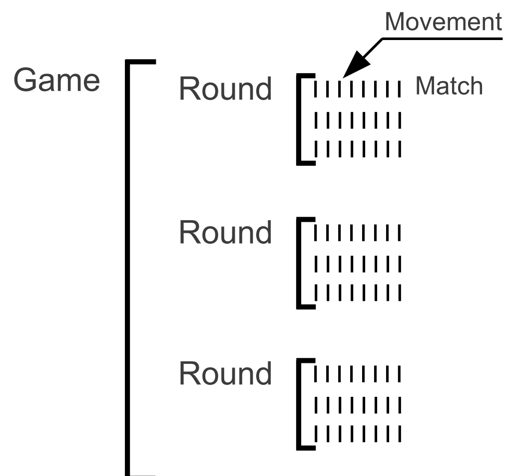

Figure 2.7: Decomposition of each game in several layers of detail

The history of the first 5 matches happens to be:

```
  (d-d)
  (c-d)
  (c-c)
  (c-c)
  (d-d)
---------
  (d-d)
  (d-d)
  (c-d)
  (d-c)
  (d-d)
---------
  (d-d)
  (d-d)
  (c-d)
  (d-d)
  (d-d)
```

```
---------
  (d-d)
  (d-d)
  (c-d)
  (c-c)
  (d-c)
---------
  (c-d)
  (c-c)
  (d-c)
  (d-d)
  (d-d)
---------
```

where each tuple represents a simultaneous move of the agents. At this moment, the game referee stops the exchange of movements, packs the history obtained so far and notifies it to the learning Agent. Our learning, agent performs the language transformation of the implicit transducer and groups letters into words. The resulting word set is

```
zxwwz
zzxyz
zzxzz
zzxwy
xwyzz
```

Note, that the learning stage occurs over the extended alphabet.

When running the ALERGIA algorithm with parameter $\alpha = 0.7$ over this training set we obtain the PDFA in Figure 2.8

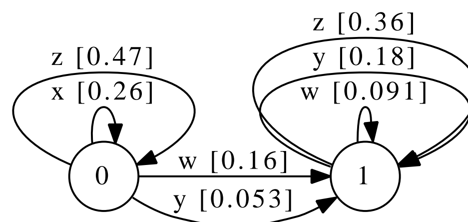

Figure 2.8: PDFA learned after the first 25 interactions

But don't forget that this automaton is representing the Mealy machine that can be seen in Figure 2.9

The cautious reader should have noticed that outgoing probability transitions in this automaton do not add 1. This is so because algorithms learning

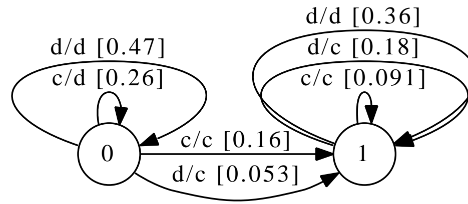

Figure 2.9: Mealy Machine derived from the learned PDFA

State	**Action**
0	c
1	d

Table 2.3: Derived actions of each state given the Mealy-Machine of Figure 2.10

PDFA take into account a transition probability to hidden sink state used to mark the end of the word. Since in our scenario there is no such concept *word* we must smooth the obtained automata in order to normalize the outgoing probabilities. The final Mealy machine can be seen in 2.10.

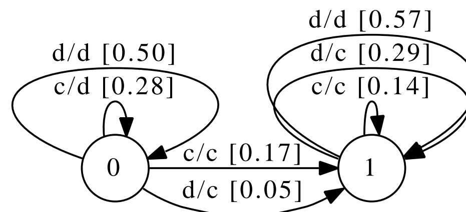

Figure 2.10: Smoothed Mealy Machine derived from the learned PDFA

Later in this thesis 4.3.1 it is shown that this probabilistic Mealy Machine corresponds with a *Markov Decision Process* for which algorithms exists that allows us to govern it optimally. Applying a policy iteration algorithm to infer our decision rules and using the discounted infinite reward function as the accumulated reward function, we obtain that for this automaton the rules governing the automaton are the ones in Table 2.3.

Chapter 3

Learning Probabilistic Finite State Automata

3.1 Introduction

The fundamental mathematical object in this thesis is a Finite State Machine, in its different flavors, deterministic, non-deterministic, probabilistic or an extended version, the transducer. It is thus necessary to explain some related concepts about languages and automata.

3.2 Symbols, Languages and Grammars

The field of Syntactic Pattern Recognition is concerned about finding structure in a stream of discrete symbols. Symbols are the elements of a finite set. Those elements are usually represented by *letters* and the set itself is called *alphabet* and is usually represented by the symbol Σ.

A *word* is a finite sequence of symbols in a given alphabet. For convenience the *empty word* is also defined and is usually represented by the greek symbol λ. The set of all the words that can be composed by a given alphabet is called the *universal language* of Σ. This language Σ^* also includes the empty word. So for example, given the alphabet consisting of a single letter $\Sigma = \{a\}$, the universal language that can be constructed is:

$$\Sigma^* = \{\lambda, a, aa, aaa, \cdots\}$$

Let's call *language* over the alphabet Σ to a subset of the universal language of Σ.

$$L \in \Sigma^*$$

We call *probabilistic or stochastic language* to the pair of a language and a function that assigns probabilities to each word of the language

$$\{L, P_L(\cdot)\}$$

3.3 Finite State Automaton

A *finite state automaton* (FSA) [HMU01, ASMO97], is defined by a five-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states
- Σ is an alphabet
- q_0 is the initial state
- $F \subseteq Q$ is the set of final states
- $\delta : Q \times \Sigma \to 2^Q$ is a partial function

We say that q is a *successor* of p if $p \in \delta(q, a)$. We call our automaton *deterministic* if for all $q \in Q$ and for all $a \in \Sigma$, the transition $\delta(q, a)$ has at most one element.

3.3.1 Quotient Automaton

Let A be a FSA and π a partition of Q, we denote by $B(q, \pi)$ as the only block that contains q and we denote the quotient set $\{B(q, \pi)|q \in Q\}$ as Q/π. Given a FSA A and a partition π over Q we define the *quotient automaton* A/π as:

$$A/\pi = (Q/\pi, \Sigma, \delta', B(q_0, \pi), \{B \in Q/\pi | B \cap F \neq \})$$

where δ' is defined as:

$$\forall B, B' \in Q/\pi,\ \forall a \in \Sigma,\ B' \in \delta'(B, a) \text{ if } \exists q, q' \in Q, q \in B, q' \in B' : q' \in \delta(q, a)$$

Given the automaton A and the partition π over Q, we have that the language defined by the quotient automaton satisfies $L(A) \subseteq L(A/\pi)$.

3.3.2 Probabilistic Automata

All the concepts explained for simple automata can be extended to the probabilistic case. A *stochastic finite automaton* (SFA), $A = (\Sigma, Q, P, q_0)$, consists of an alphabet Σ, a finite set of nodes $Q = \{q_0, q_1, \ldots, q_n\}$ with q_0 the initial node, and a set P of probability matrices $p_{ij}(a)$ giving the probability of a transition from node q_i to node q_j led by the symbol a in the alphabet. If we call p_{if} the probability that the string ends at node q_i, the following constraint applies:

$$p_{if} + \sum_{q_j \in Q} \sum_{a \in A} p_{ij}(a) = 1$$

The probability $p(w)$ for the string w to be generated by A is defined by:

$$p(w) = \sum_{q_j \in Q} p_{0j}(w) p_{jf}$$
$$p_{ij}(w) = \sum_{q_k \in Q} \sum_{a \in A} p_{ik}(wa^{-1}) p_{kj}(a)$$

and the language generated by the automaton A is defined as:

$$L = \{w \in \Sigma^* : p(w) \neq 0\}$$

Those languages generated by means of a SFA are called *stochastic regular laguages*. In case the SFA contains no *useless nodes*[1], it generates a probability distribution for the strings in Σ^*:

$$\sum_{w \in \Sigma^*} p(w) = 1$$

The algorithms that we are going to use to infer the probabilistic automata, are restricted to learn *probabilistic deterministic finite automata* (PDFA). This means that for every node $q_i \in Q$ and symbol $a \in A$ there exists at most one node such $p_{ij} \neq 0$. In such cases a transition function $h = \delta(i, a)$ can be defined.

3.4 Algorithms

3.4.1 Identifying regular languages

Here we present some concepts that will be useful when developing later the automata algorithms.

A **prefix tree acceptor** (PTA) is a tree-like DFA built from the learning sample by taking all the prefixes in the sample as states and constructing the smallest DFA which is a tree. A formal algorithm *buildPrefixTree* can be seen in Algorithm 1. Note that we can also build a PTA from a set of positive strings only. This corresponds to building the PTA($\langle S_+, \emptyset \rangle$).

The algorithm we are going to study takes the PTA as a starting point and tries to generalise from it by merging states. In order not to get lost in the process it will be interesting to divide the states into three categories:

1. The RED states which correspond to states that have been analysed and which will not be revisited; they will be the states of the final automaton

2. The BLUE states which are the **candidate** states: they have not been analysed yet and it should be from this set that a state is drawn in order to consider merging it with a RED state

[1] A node q_i is useless if there are no strings $x, y \in \Sigma^*$ such that $\sum_j p_{1i}(x) p_{ij}(y) p_{jf} \neq 0$

Algorithm 1 *buildPrefixTree*

```
Input: A sample ⟨S+, S−⟩
Output: A = PTA(⟨S+, S−⟩) = ⟨Σ, Q, qλ, FA, FR, δ⟩
FA ← ∅
FR ← ∅
Q ← {qu : u ∈ PREF(S+ ∪ S−)}
for qu·a ∈ Q do
  δ(qu, a) ← qua
end for
for qu ∈ Q do
  if u ∈ S+ then
    FA ← FA ∪ qu
  end if
  if u ∈ S− then
    FR ← FR ∪ qu
  end if
end for
return A
```

3. The WHITE states, which are all the others. They will in turn become BLUE and then RED

The basic operations that allow the manipulation of the PTA are **compatible, merge** and **promote**.

compatible: deciding equivalence between states

The question here is of deciding if two states are compatible or not, that is, if merging these two states will not result in creating confusion between accepting and rejecting states. Typically the compatibility might be tested by:

$$q \simeq_A q' \iff L_{F_A}(A_q) \cap L_{F_R}(A_{q'}) = \emptyset \text{ and } L_{F_R}(A_q) \cap L_{F_A}(A_{q'}) = \emptyset$$

But it happens that the formula above is not sufficient to merge two states and there are situations in which more operations like merging, folding and then testing consistency are needed.

merge: merging two states

The merging operation takes two states from an automaton and merges them into a single state. It should be noted that the effect of the merge is that a deterministic automaton will probably lose the determinism property through this operation, and thus we will attempt to avoid having to use these automata.

promote: promoting a state

Promotion is another deterministic and greedy decision. The idea here is that having decided that a state in the PTA that is a candidate for merging with the final automata state, this state should finally become a final state of the resulting automata and should not be merged.

3.4.2 RPNI

Although not an algorithm to learn Probabilistic DFAs but regular Deterministic Finite Automata, the algorithm RPNI [Onc92] is the base for the Alergia (3.4.3) and subsequently the MDI algorithm (3.4.4). It is thus interesting to briefly review how it works.

Although its similarities with its probabilistic derivations, this algorithm needs both a sample of positive attributes (S_+) belonging to the language we are trying to learn and a set of negative examples (S_-) that do not belong to the intended language. Obviously, the more examples both positive and negative, the more is covered the language and thus the easier to learn exactly.

This algorithm starts by building the prefix tree acceptor of the positive instances of the training sample (S_+) and then proceeds by iteratively choosing possible merges, checks if a given merge is correct and is made between two compatible states, makes the merge if admissible and promotes the state if no merge is possible.

The algorithm has as a starting point the PTA, which is a deterministic finite automaton. In order to avoid problems with non-determinism, the merge of two states is immediately followed by a folding operation: the merge in RPNI always occurs between a state already selected as final and a state that is considered in the iteration.

At the end of the process we expect the obtained automaton to accept the strings present in the training sample and to reject the negative ones.

Algorithm 2 RPNI-PROMOTE

Input: a DFA $A = \langle \Sigma, Q, q_\lambda, F_A, F_R, \delta \rangle$, sets RED,BLUE $\subseteq Q, q_u \in$ BLUE
Output: A, RED, BLUE updated
RED $\leftarrow$ RED $\cup \{q_u\}$
BLUE $\leftarrow$ BLUE $\cup \{\delta(q_u, a), a \in \Sigma\}$
return A, RED,BLUE

Example

Here we show how a PTA is built from a set of examples. For this dataset we have that $S_+ = \{011, 101\}$ and $S_- = \{1, 01\}$. So the PTA obtained from the set of positive examples can be seen in Figure 3.1.

Algorithm 3 RPNI-COMPATIBLE

```
Input: A, S_−
Output: a Boolean, indicating if A is consistent with S_−
for w ∈ S_− do
   if δ_A(q_λ, w) ∩ F_A ≠ ∅ then
      return False
   end if
end for
return True
```

Algorithm 4 RPNI-MERGE

```
Input: a DFA A, states q ∈ Red, q′ ∈ Blue
Output: A updated
Let (q_f, a) be such that δ_A(q_f, a) = q′
δ_A(q_f, a) ← q
return RPNI-FOLD(A, q, q′)
```

Algorithm 5 RPNI-FOLD

```
Input: a DFA A, states q, q′ ∈ Q q' being the root of a tree
Output: A updated, where subtree q′ is folded into q
if q′ ∈ F_A then
   F_A ← F_A ∪ {q}
end if
for a ∈ Σ do
   if δ_A(q′, a) is defined then
      if δ_A(q, a) is defined then
         A ← RPNI-FOLD(A, δ_A(q, a), δ_A(q′, a))
      else
         δ_A(q, a) ← δ_A(q′, a)
      end if
   end if
end for
```

Algorithm 6 RPNI

Input: a sample $S = \langle S_+, S_- \rangle$, functions COMPATIBLE, CHOOSE
Output: a DFA $A = \langle \Sigma, Q, q_\lambda, F_A, F_R, \delta \rangle$
$A \leftarrow$ BUILD-PTA(S_+)
RED $\leftarrow \{q_\lambda\}$
BLUE $\leftarrow \{q_a : a \in \Sigma \cap \text{PREF}(S_+)\}$
while BLUE $\neq \emptyset$ **do**
  CHOOSE($q_b \in$ BLUE)
  BLUE $\leftarrow$ BLUE $\setminus \{q_b\}$
  if $\exists q_r \in$ RED such that RPNI-COMPATIBLE(RPNI-MERGE(A, q_r, q_b), S_-) **then**
    $A \leftarrow$ RPNI-MERGE(A, q_r, q_b)
    BLUE $\leftarrow$ BLUE $\cup \{\delta(q,a) : q \in$ RED $\wedge\, a \in \Sigma \wedge \delta(q,a) \notin$ RED$\}$
  else
    $A \leftarrow$ RPNI-PROMOTE(q_b, A)
  end if
end while
for $q_r \in$ RED **do**
  if $\lambda \in (L(A_{q_r}))^{-1} S_-$ **then**
    $F_R \leftarrow F_R \cup \{q_r\}$
  end if
end for

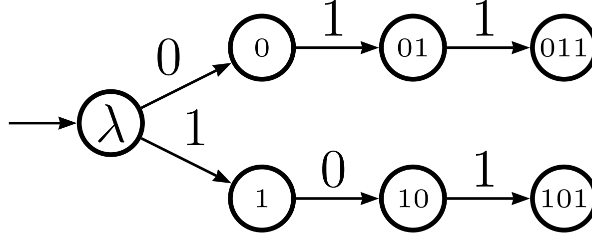

Figure 3.1: Prefix tree of the positive training sample

3.4.3 ALERGIA

The ALERGIA algorithm [CO94] for learning PDFAs follows the same principles than the RPNI algorithm seen in Section 3.4.2. First begins by building the Prefix Tree Acceptor (PTA) from the training sample and evaluates at every node the relative probabilities of the transitions coming out from the node. Next it tries to merge couples of nodes following a well defined order (essentially, that of the levels in the PTA or lexicographical order). Merging is performed if the resulting automaton is, within statistical uncertainty, equivalent to the PTA. The process ends when further merging is not possible. The

algorithm can be seen in Algorithm 9

Algorithm 7 ALERGIA-TEST

Input: an FFA $A, f_1, n_1, f_2, n_2, \alpha > 0$
Output: a Boolean indicating if the frecuencies $\frac{f_1}{n_1}$ and $\frac{f_2}{n_2}$ are sufficiently close
$\gamma \leftarrow \frac{f_1}{n_1} - \frac{f_2}{n_2}$
return $\left(\gamma < \left(\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}}\right) \cdot \sqrt{\frac{1}{2} \ln \frac{2}{\alpha}}\right)$

The compatibility test makes use of the Hoeffding bounds. The algorithm ALERGIA-COMPATIBLE (8) calls the ALERGIA-TEST (7) as many times as needed, this number being finite due to the fact that the recursive calls visit a tree.

The basic function CHOOSE is as follows: take the smallest state in an ordering that has been done at the beginning (on the PTA). The test that is used to decide if the states are to be merged or not (function COMPATIBLE) is based on the Hoeffding test made on the relative frequencies of the empty string and of each prefix.

Algorithm 8 ALERGIA-COMPATIBLE

Input: an FFA A, two states $q_u, q_v, \alpha > 0$
Output: q_u and q_v compatible?
$Correct \leftarrow True$
if ALERGIA-TEST$(F_{P\ A}(q_u), \mathrm{FREQ}_A(q_u), F_P\ A(q_v), \alpha)$ **then**
  $Correct \leftarrow False$
end if
for $a \in \Sigma$ **do**
  if ALERGIA-TEST$(\delta_{fr}(q_u, a), \mathrm{FREQ}_A(q_u), \delta_{fr}(q_v, a), \mathrm{FREQ}_A(q_v), \alpha)$ **then**
    $Correct \leftarrow False$
  end if
end for

3.4.4 MDI

Algorithm ALERGIA decided upon merging (and thus generalisation) through a local test: substring frequencies are compared and if it is not unreasonable to merge, then merging takes place. A more pragmatic point of view could be to merge whenever doing so is going to give us an advantage. The goal is of course to reduce the size of the hypothesis while keeping the predictive qualities of the hypothesis (at least with respect to the learning sample) as good as possible. For this we can use the likelihood of each string. The goal is to obtain a good balance between the gain in size and the loss in perplexity.

Algorithm 9 ALERGIA

Input: a sample S, $\alpha > 0$
Output: an FFA A
Compute t_0, threshold on the size of the multiset needed for the test to be statistically significant
$A \leftarrow PTA(S)$
$\textsc{Red} \leftarrow \{q_\lambda\}$
$\textsc{Blue} \leftarrow \{q_a : a \in \Sigma \cap \textsc{Pref}(S)\}$
while CHOOSE(q_b) from $\textsc{Blue}$ such that FREQ(q_b) $\geq t_0$ **do**
  if $\exists q_r \in \textsc{Red}$: ALERGIA-COMPATIBLE(A, q_r, q_b, α) **then**
    $A \leftarrow$ STOCHASTIC-MERGE(A, q_r, q_b)
  else
    $\textsc{Red} \leftarrow \textsc{Red} \cup \{q_b\}$
  end if
  $\textsc{Blue} \leftarrow \{q_{ua} : ua \in \textsc{Pref}(S) \wedge q_u \in \textsc{Red}\} \setminus \textsc{Red}$
end while
return A

Attempting to find a good compromise between these two values is the main idea of algorithm MDI (Minimum Divergence Inference).

Algorithm 10 MDI-COMPATIBLE

Input: an FFA A, two states q and $q', S, \alpha > 0$
Output: a Boolean indicating if q and q' are compatible
$B \leftarrow$ STOCHASTIC-MERGE(A, q, q')
return ($score(S, B) < \alpha$)

The key difference is that the recursive merges are made inside Algorithm MDI-COMPATIBLE (3.4.4) and before the new score is computed instead of in the main algorithm.

Like in the ALERGIA algorithm, a difficult question to answer is that of setting the tuning parameter (α): if set too high, merges will take place early, which will perhaps include a wrong merge, prohibiting later necessary merges, and the result can be bad. On the contrary, a small α will block all merges, including those that should take place, at least until there is little data left. This is the *safe* option, which leads in most cases to very little generalization.

Algorithm 11 MDI

Input: a sample $S, \alpha > 0$
Output: an FFA A
Compute t_0, the threshold on the size of the multiset needed for the test to be statistically significant
$A \leftarrow PTA(S)$
RED $\leftarrow \{q_\lambda\}$
BLUE $\leftarrow \{q_a : a \in \text{PREF}(S)\}$
$current_score \leftarrow score(S, PTA(S))$
while CHOOSE(q_b) from BLUE such that FREQ(q_b) $\geq t_0$ **do**
  if $\exists q_r \in$ RED : MDI-COMPATIBLE(A, q_r, q_b, S, α) **then**
    $A \leftarrow$ STOCHASTIC-MERGE(A, q_r, q_b)
  else
    RED $\leftarrow$ RED $\cup \{q_b\}$
  end if
  BLUE $\leftarrow \{q_{ua} : ua \in \text{PREF}(S) \wedge q_u \in \text{RED}\} \setminus$ RED
end while
return A

Chapter 4

Opponent Modelling

4.1 Defeating an Opponent

In Chapter 3 we learned about the algorithms used for inferring the PDFAs. Those algorithms are used in the context of a repeated game setup where our learning agent is trying to defeat an opponent. This chapter is about the two main steps that must be performed to accomplish this goal:

1. Derive a working hypothesis about the internal strategy that our opponent is using. This will be accomplished using the GIATI algorithm.

2. Derive a counter-strategy that could exploit any weaknesses that strategy could have. This will be accomplished translating our working hypothesis into a Markov Decision Process and deriving the governing rules for it.

4.2 The GIATI algorithm

In the previous chapter we have seen techniques to infer probabilistic automata from examples but in this thesis what we are trying to infer is the transducer that our opponent is using to play the game. So, we should fill the gap between having algorithms that learn PDFAs and to convert those models into the target transducer. This chapter is devoted to the necessary techniques to accomplish this task.

4.2.1 Some terminology

A finite-state transducer (FST), T, is a tuple $\langle \Sigma, \Delta, Q, q_0, F, \delta \rangle$, in which:

1. Σ is a finite set of source symbols

2. Δ is a finite set of target symbols

3. Q is finite set of states

4. q_0 is the initial state

5. $F \subseteq Q$ is a set of final states

6. $\delta \subseteq Q \times \Sigma \times \Delta^* \times Q$ is a set of transitions.

Note that $\Sigma \cap \Delta = \emptyset$. A *translation form* ϕ of length I in T is defined as the sequence of transitions:

$$\phi = (q_0^\phi, s_1^\phi, t_1^\phi, q_1^\phi)(q_1^\phi, s_2^\phi, t_2^\phi, q_2^\phi) \cdots (q_{I-1}^\phi, s_I^\phi, t_I^\phi, q_I^\phi)$$

where $(q_{i-1}^\phi, s_i^\phi, t_i^\phi, q_i^\phi) \in \delta$. A pair $(s,t) \in \Sigma^* \times \Delta^*$ is a **translation pair** if there is a translation form ϕ of length I in T such that $I = |s|$ and $t = t_1^\phi t_2^\phi \cdots t_I^\phi$. A **rational translation** is the set of all translation pairs of some finite-state transducer T.

This definition of a finite-state transducer is similar to the definition of a regular or finite-state grammar. The main difference is that in a finite-state grammar, the set of target symbols Δ does not exist, and the transitions are defined on $Q \times \Sigma \times Q$. A translation form is the transducer counterpart of a derivation in a finite-state grammar, and the concept of rational translation is reminiscent of the concept of (regular) language, defined as the set of strings associated with the derivations in the grammar G.

A *stochastic finite-state transducer* , T_P is defined as a tuple $\langle \Sigma, \Delta, Q, q_0, p, f \rangle$ in which Q, q_0, Δ, Σ are as in the definition of a finite-state transducer and p and f are two functions:

1. $p : Q \times \Sigma \times \Delta^* \times Q \to [0,1]$

2. $f : Q \to [0,1]$

That satisfy $\forall q \in Q$

$$f(p) + \sum_{(a,w,q') \in \Sigma \times \Delta^* \times Q} p(q,a,w,q') = 1$$

The **probability of a translation pair** $(s,t) \in \Sigma^* \times \Delta^*$ according to T_P is the sum of the probabilities of all the translation forms of (s,t) in T:

$$P_{T_P}(s,t) = \sum_{\phi \in d(s,t)} P_{T_P}(\phi)$$

where the probability of a translation form ϕ is

$$P_{T_P}(\phi) = \prod_{i=0}^{I} p(q_{i-1}, s_i, t_i, q_i) \cdot f(q_I)$$

that is, the product of the probabilities of all the transitions involved in ϕ.

There are two main types of finite state transducers, the Moore machine and the Mealy machine. Since we are interested in its stochastic derivations, we will show the definition of the probabilistic instances. The definition of the deterministic entities could be obtained by simply taking into account degenerate probabilities where only one transition has the full weight, i.e. there is a transition that has probability 1.

Moore machine

A Moore machine is a deterministic automaton with the ability to generate symbols. Like other automata, the Moore machine performs state transitions depending on the input symbol consumed. When the automata lands in a new state, an output symbol is generated according to an internal formula.

Formally a stochastic Moore machine is a tuple $\langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$ where:

- Q is the set of nodes in the automaton
- Σ and Δ are the input and output alphabets
- $\delta : Q \times \Sigma \to P_Q$ is the set of probability distributions over Q
- $\lambda : Q \to P_\Delta$ is the probabilistic *output* function. The Moore machine generates output symbols according to a given probability function P_Δ
- q_0 is the initial state

Mealy machine

A Mealy machine is also a deterministic automaton that generates output symbols *during* the state transition. The main difference with the Moore machine is that you can define different probability functions for the transitions and for the output symbols while in the Mealy machine the probability function is a joint function for the transitions and the output symbols.

Mathematically a Mealy machine is a tuple $\langle Q, \Sigma, \Delta, \delta, q_0 \rangle$ where:

- Q is the set of nodes in the automata
- Σ and Δ are the input and output alphabets
- $\delta : Q \times \Sigma \to P_{Q\times\Delta}$ is the set of probability distributions over the set of transitions and output symbols $Q \times \Delta$
- q_0 is the initial state

The GIATI algorithm relies on the following two theorems. The interested reader should check [Ber09] for the corresponding proofs.

Theorem 4.1. *$T \subseteq \Sigma^* \times \Delta^*$ is a rational translation if and only if there exists an alphabet Γ, a regular language $L \subset \Gamma^*$, and two morphisms $h_\Sigma : \Gamma^* \to \Sigma^*$ and $h_\Delta : \Gamma^* \to \Delta^*$ such that $T = \{(h_\Sigma(w), h_\Delta(w)) | w \in L\}$.*

and

Theorem 4.2. *A distribution $P_T : \Sigma^* \times \Delta^* \to [0,1]$ is a stochastic rational translation if and only if there exist an alphabet Γ, two morphism $h_\Sigma : \Gamma^* \to \Sigma^*$ and $h_\Delta : \Gamma^* \to \Delta^*$, and a stochastic regular language P_L such that $\forall (s,t) \in \Sigma^* \times \Delta^*$,*

$$P_T(s,t) = \sum_{w \in \Gamma^* : (h_\Sigma(w), h_\Delta(w)) = (s,t)} P_L(w)$$

4.2.2 Inferring Finite-State Transducers

The methodology explained in [CV04] is called **grammatical inference and alignment for transducer inference** (GIATI). Based on the works of Berstel [Ber09] it is well known that (stochastic) rational translation T can be obtained as a homomorphic image of certain (stochastic) regular language L over an adequate alphabet Γ.

This suggest the following general technique for learning a stochastic finite-state transducer, given a finite sample I_+ of string pairs $(s,t) \in \Sigma^* \times \Delta^*$:

1. Each training pair (s,t) from I_+ is transformed into a string $\mathbf{z}$ from an extended alphabet Γ (strings of Γ-symbols) yielding a sample $\mathcal{S}$ of strings $\mathcal{S} \subset \Gamma^*$. Lets call this transformation $\mathcal{L} : \Sigma^* \times \Delta^* \to \Gamma^*$

2. A (stochastic) regular grammar $\mathcal{G}$ is inferred from $\mathcal{S}$

3. The Γ-symbols of the grammar rules are transformed back into pairs of source/target symbols/strings (from $\Sigma^* \times \Delta^*$). The "inverse labelling function" $\Lambda : \Gamma^* \to \Sigma^* \times \Delta^*$ is one that $\Lambda(\mathcal{L}(I_+)) = I_+$. Following Theorems 4.2 and 4.1, $\Lambda(\cdot)$ consists of a couple of morphisms, h_Σ, h_Δ, such that for a string $z \in \Gamma^*$, $\Lambda(z) = (h_\Sigma(z), h_\Delta(z))$

The overall procedure can be seen in the Figure 4.1.

$$\begin{array}{ccc} A \subset \Sigma^* \times \Delta^* & \xrightarrow{\text{Labeling - } \mathcal{L}(\cdot)} & S \subset \Gamma^* \\ \downarrow & & \text{GI} \downarrow \text{algorithm} \\ \mathcal{T} : A \subset T(\mathcal{T}) & \xleftarrow{\text{Inverse labeling - } \Lambda(\cdot)} & \mathcal{G} : S \subset L(\mathcal{G}) \end{array}$$

Figure 4.1: Commutative diagram of the transformations performed using the GIATI algorithm

4.2.3 Applying GIATI in the repeated games scenario

When applying the GIATI algorithm three placeholders have to be filled in order to run the algorithm, the labelling function $\mathcal{L}(\cdot)$, the inverse labelling function $\Lambda(\cdot)$ and the algorithm used to learn the probabilistic automaton. For our opponent modelling scenario that has been defined in Section 2.3, the GIATI elements are defined as follows:

- Choosing the labelling function $\mathcal{L}$ tends to be a difficult decision. In statistical translation, that is one of the domains where FST are of common use, a previous study of the statistical correlations between the positions of the words in the source language and the target language, called statistical alignment, is performed. In our case, since the action and the response are always paired, there is no need to perform such statistical analysis of the data, each input symbol is paired with only one output symbol.

  So, the labelling function $\mathcal{L}$ will be

  $$\mathcal{L}(s,t) = concat(s,t) = st = z \qquad z \in \Gamma$$

  where the symbol $st \in \Gamma$ is obtained from the alphabet Γ that receives its elements from all the permutations of the symbols in Σ and Δ.

- Learning the probabilistic automata will be performed using one of the before mentioned algorithms, ALERGIA and MDI

- The inverse labelling function just splits the source symbol from the target symbol

  $$\Lambda(z) = \Lambda(st) = (s,t) \qquad s \in \Sigma, t \in \Delta$$

4.3 Markov Decision Processes

A Markov Decision Process (MDP) [Ber95, Put94] consists of five elements: decision epochs, states, actions, transition probabilities, and rewards.

Decision Epochs and Periods

Decisions are made at points of time referred as *decision epochs*. Let T denote the set of decision epochs, in a discrete environment T can be finite or infinite ranging in the real positive line. In our discrete environment, at each epoch decisions are made to govern the probabilistic system.

State and Action Sets

At each decision epoch, the system occupies a *state*. We denote the set of possible system states as S. If, at some decision epoch, the decision maker observes the system in state $s \in S$, he may choose action a from the set of allowable actions in state s, A_s. Given this setup let $A = \cup_{s \in S} A_s$. Note that S and A do not vary with t.

Actions may be chosen either randomly or deterministically. Choosing actions randomly means selecting a probability distribution $q(\cdot) \in P(A_s)$ being $P(A_s)$ the collection of probability distributions on subsets of A_s. When we are dealing with deterministic selection of actions, our model is simply using degenerate probability distributions.

Rewards and Transition Probabilities

As a result of choosing action $a \in A_s$ in state s,

1. the decision maker receives a reward, $r_t(s, a)$ and
2. the system state at the next decision epoch is determined by the probability distribution $p(\cdot|s, a)$

From the perspective of the models, it is immaterial how the reward is accrued during the period. We only require that its value or *expected value* be known before choosing an action, and that it not be effected by future actions. The reward might be,

1. a lump sum received prior to the next decision epoch
2. a random quantity that depends on the system state at the subsequent epoch
3. or a combination of the above

When the reward depends on the state of the system at the next decision epoch, we let $r(s, a, j)$ denote the value at time t of the reward received when the sttate of the system at decision epoch t is s, action $a \in A_s$ is selected and the system occupies state j at decision epoch $t + 1$. Its expected value at decision epoch t may be evaluated by computing

$$r_t(s, a) = \sum_{j \in S} r_t(s, a, j) p_t(j|s, a)$$

We usually assume that

$$\sum_{j \in S} p_t(j|s, a) = 1$$

We refer to the collection of objects forming a tuple

$$\langle T, S, A_s, p(\cdot|s, a), r(s, a, j) \rangle \tag{4.1}$$

as a *Markov Decision Process*. The qualifier "Markov" is used because the transition probabilities and reward functions depend on the past only through the current state of the system and the action selected by the decision maker in that state.

Decision Rules

A *decision rule* prescribes a procedure for action selection in each state at a specified decision epoch. There is a lot of variety on how decision rules pick their actions but the primary focus in this thesis will be on deterministic Markovian rules because they are easier to implement and evaluate. Such decision rules are functions $d : S \to A_s$, which specify the action choice when the system occupies state s, and thus for each $s \in S$ we have $d(s) \in A_s$. This decision rule is said to be *Markovian* (memoryless) because it does not depends on previous system states and actions only thought the current state of the system, and *deterministic* because it chooses an action with certainty.

Policies

A *policy* specifies the decision rule to be used at all decision epochs. We call a policy *stationary* if $d_t = d$ for all $t \in T$

4.3.1 Links between Probabilistic Mealy Machines and MDP

The GIATI algorithm provides us with a transducer in the form of a non-deterministic Mealy Machine. There is an almost direct translation between this Mealy machine and an MDP but some considerations should be taken into account. Lets recap the definitions of those mathematical objects, a Mealy Machine (Section 4.2.1) is the tuple:

$$\langle Q, \Sigma, \Delta, \delta, q_0 \rangle$$

and a Markov Decision Process (Equation 4.1) is defined by:

$$\langle T, S, A_s, p(\cdot|s, a), r(s, a, s') \rangle$$

So the different transformations can be summarized by

- The decision epochs T in a MDP are more general than the step transitions in a transducer since in an MDP, continuous or Borel sets are allowed as epoch. Since we are doing the translation from a Mealy machine to an MDP the translation is direct, the MDP is fixed to discrete and evenly spaced decision epochs and no further adaptations must be done.

- The set of states Q and S can be used interchangeably between the two models. Here we perform a direct translation
- For the transition probabilities in an MDP, $p(s'|s,a)$ some derivations must be performed. In a Mealy machine our transition function δ specifies probabilities for the pairs next state plus emitting symbol, $Q \times \Delta$, so the basic translation should be

$$p(s'|s,a) = \sum_{t \in \Delta} p(s,a,s',t)$$

- The reward function, $r(s,a,s')$ depends on the game we are playing. When a transition happens in a Mealy machine, both the input symbol and the emitted symbol are available. This represents a movement that a referee later will assign a value to each player, so in order to provide an expected value for a transition, the rewards are weighted by the probabilities of a given transition happening when receiving an input symbol

$$r(s,a,s') = \sum_{t \in \Delta} p(s,a,s',t) \cdot evalMove(a,t)$$

  Where the function *evalMove* is game dependent.

With these simple translations, a given Mealy machine obtained by the GIATI algorithm can be translated into a Markov Decision Process, ready for obtaining the governing decision rules.

4.3.2 Infinite horizon models

When a Markov decision process is run indefinitely, each policy induces a discrete-time reward process. That reward stream has an associated utility to the agent depending on some functions used to value that stream. Several functions can be used to aggregate that stream into a single value for easiness when comparing different streams:

1. The *expected total reward policy* π, v^π is defined to be

$$v^\pi(s) \equiv \lim_{N \to \infty} E_s^\pi \left\{ \sum_{t=1}^{N} r(X_t, Y_t) \right\} = \lim_{N \to \infty} v_{N+1}^\pi(s) \tag{4.2}$$

   Note that the limit in 4.2 may be $+\infty$ or $-\infty$ and consequently this performance measure is not always appropriate.

2. The *expected discounted reward* of policy π is defined to be

$$v_\lambda^\pi(s) \equiv \lim_{N \to \infty} E_s^\pi \left\{ \sum_{t=1}^{N} \lambda^{t-1} r(X_t, Y_t) \right\}$$

   for $0 \leq \lambda \leq 1$

3. The *average reward* or *gain* of policy π is defined by

$$g^{\pi}(s) \equiv \lim_{N\to\infty} \frac{1}{N} E_s^{\pi} \left\{ \sum_{t=1}^{N} r(X_t, Y_t) \right\}$$

The expected total discounted reward criterion

The discounted reward model is the function valuation that will be employed in this thesis. The discounted reward has strong links with the economic behaviour and decision theoretic literature and our learning agent is dealing with utility functions to determine which actions to take. Moreover, discounted functions present some mathematical niceties that allow easy function usage, we do not need to worry about the convergence of the reward series.

Discounted policy also helps in our game setup since our agent assigns greater value to rewards closer in time. Although games are not infinite, the agent does not know how long they are going to take and thus a strategy that assigns uncertainty to distant and future rewards is preferable.

4.3.3 Finding optimal policies

The transitions in a Markov Decision Process are independent of the transition history that took the MDP to a given state and by derivation, it has no dependency either on the decision that generated the transitions in the first place. So the objective is to find a collection of pair state/action, $\pi = \{(q_0, a_0), (q_1, a_1), \cdots, (q_S, a_S)$, that defines our policy.

Given a Markov Decision Process we are interested on a policy π_{λ}^{π}

$$v_{\lambda}^{\pi^*} = \sup_{\pi} v_{\lambda}^{\pi}(s)$$

where the expected total discounted reward is defined by

$$v_{\lambda}^{\pi}(s) = E_s^{\pi} \left\{ \sum_{t=1}^{\infty} \lambda^{t-1} r(q_t, a_t) \right\}$$

Several algorithms exist for finding such policies like *value iteration*, *policy iteration*, *modified policy iteration* and *linear programming* [Put94, Ber95]. In this thesis we will show the two most common procedures *value iteration* and *policy iteration*.

Value iteration

The main idea behind the *value iteration* is that the utility of a given state is computed iteratively. The procedure guarantees that in the limit, the utility converges to its real value. In practice the algorithm does not go to the infinity but instead stops when the utility does not change above a specified threshold.

In Algorithm 12 the value iteration algorithm can be seen.

Algorithm 12 Value Iteration Algorithm

Input: an MDP M, a threshold ε
Output: a function $d : states \times actions$
$v^0 \leftarrow 0, n \leftarrow 0, S \leftarrow states(M)$
repeat
  for all $s \in S$ **do**
    $v^{n+1}(s) = \max_{a \in A_s} \left\{ r(s,a) + \sum_{j \in S} \lambda p(j|s,a) v^n(j) \right\}$
  end for
until $||v^{n+1} - v^n|| < \varepsilon(1-\lambda)/2\lambda$
return $d_\varepsilon(s) \in \arg\max_{a \in A_s} \left\{ r(s,a) + \sum_{j \in S} \lambda p(j|s,a) v^{n+1}(j) \right\}$

Policy iteration

The *value iteration algorithm* first computes the set of utility functions for each state and when those are calculated, it chooses the action that gives best utility overall. While value iteration can be regarded as a fixed point algorithm, *policy iteration* relates directly to the structure of Markov Decision Processes. In the policy iteration algorithm, a set of rules is iterated until no change in the rules provides better performance. It is known that this procedure provides better performance and faster convergence rates.

4.4 Summary

Let's briefly summarize the steps that our agent takes in order to play the games.

First it will play a round of games choosing random movements because that is the best strategy that can be followed when you have no idea about the opponent's strategy.

After this first round, all the pairs of movements will be collected. Those pairs will be transformed to single symbols of an extended alphabet as is explained in the GIATI algorithm.

With the new set of symbols and the associated words, a PDFA is learned. Once we have this automaton a Mealy Machine is derived using the function to revert symbols from the extended alphabet to pairs of input-output symbols.

The Mealy Machine is then converted to a Markov Decision Process and the Policy Iteration algorithm is used to derive a set of rules. The rules are a dictionary with the action that has to be performed when the Mealy Machine is in a given state.

With the new Mealy Machine and the set of rules the agent plays another round of movements.

Algorithm 13 Policy Iteration

Input: a MDP M
Output: a policy π with an action for each state in M
$V(s) \in \Re$ and $\pi(s) \in A(s)$ randomly chosen for all $s \in states(M)$
repeat
  // Policy evaluation
  $\Delta \leftarrow \infty$
  repeat
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s'} P_{ss'}^{\pi(s)}[R_{ss'}^{\pi(s)} + \gamma V(s')]$
    $\Delta \leftarrow \min(\Delta, |v - V(s)|)$
  until $\Delta < \epsilon$
  // Policy improvement
  $policyStable \leftarrow true$
  for all $s \in S$ **do**
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} P_{ss'}^{\pi(s)}[R_{ss'}^{\pi(s)} + \gamma V(s')]$
    if $b \neq \pi(s)$ **then**
      $policyStable \leftarrow false$
    end if
  end for
until $policyStable = true$
return π

Chapter 5

Experiments

In the previous chapters, we have delineated the framework that uses PDFA learning algorithms for controlling an opposing agent in a game setup. How well this algorithmic platform works in the task of winning an opponent is something we will study here.

5.1 The experiments

Here we describe the experimental setup we have used. All the graphs contain the results for each of the algorithms under study (ALERGIA and MDI) plus the results of using a random player against the opponent. The other measure taken is how much could we made against the opponent if we knew on each time step in which state the opponent is and issuing the command most favourable to us.

The software

In order to conduct the experiments that follow we developed a testing framework where we could play with different automata and algorithm configuration. The code for the ALERGIA algorithm and the MDI were kindly provided by the authors [Onc92, Tho00].

5.2 Prisoner's Dilemma

For the study of the Prisoner's dilemma game we will use as opponents three different Moore Machine borrowed from the famous tournament held by Robert Axelrod [Axe84]. In the original tournament, fifteen attendees competed in round-robin tournament of the iterated Prisoner Dilemma, where every interaction was based on 200 repetitions of the PD game. In the study we are going to use three of the best opponents sent to the tournament. Those opponents can be seen in Figure 5.1. The automata in the figure must be regarded as Moore Machines. Here we haven't explicitly stated the emitting symbols and

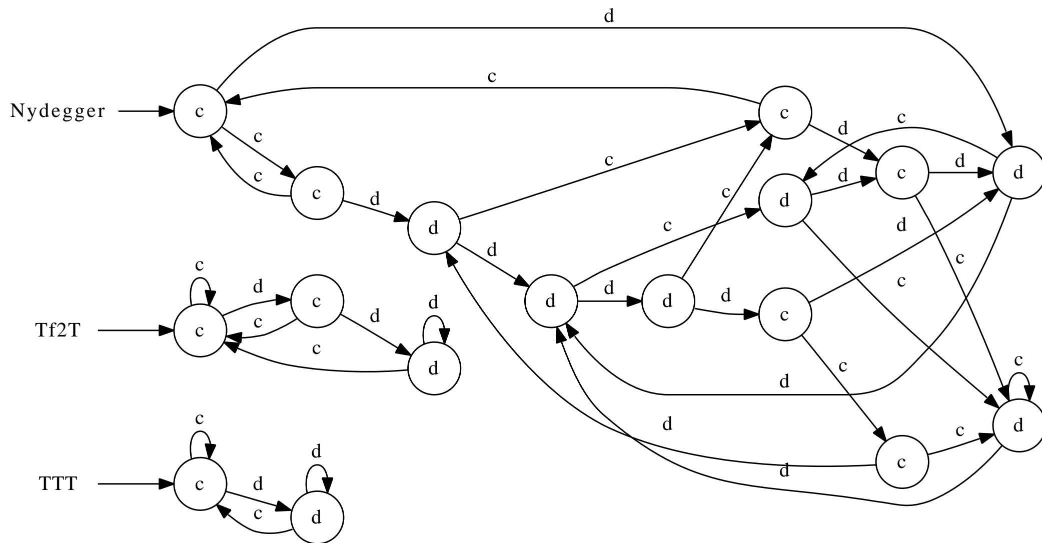

Figure 5.1: Prisoner's Dilemma opponents used in the study

probabilities in order to not clutter the diagram. The labels on each state must be regarded as the prevailing emitting symbol, that is, a **c** represents that a state has probabilities $P_{state}(c) = 0.9$ and $P_{state}(d) = 0.1$, and conversely, a **d** represents $P_{state}(c) = 0.1$ and $P_{state}(d) = 0.9$. The exact numbers of the probabilities were chosen in order to introduce a meaningful probabilistic behaviour in the automata keeping at the same time its original intended purpose due to its architecture. Here are a brief description of the meaning of each of the automata:

- **Tit-for-Tat** (TTT): Cooperates at the first iteration and then follows (most likely) the previous opponent's action.
- **Tit-for-two-Tat: Tf2T** Cooperates (most likely) for the two first states and defects (most likely) after two consecutive defections of its opponent.
- **Nydegger** Behaves like TTT at the beginning and then behaves according to the three previous joint actions of both players.

5.2.1 Round length

In this set experiments we want to study the effect of the round length in the game outcome. On each round a new agent hypothesis is created with the movements of this round. So the best agent that we could obtain would be the one learnt after the full game happens, but this would leave us with the random player playing all the movements in the beginning and we are primarily interested in wining the game. So a balance or trade-off has to be found for better performance. This section will study how increasing the

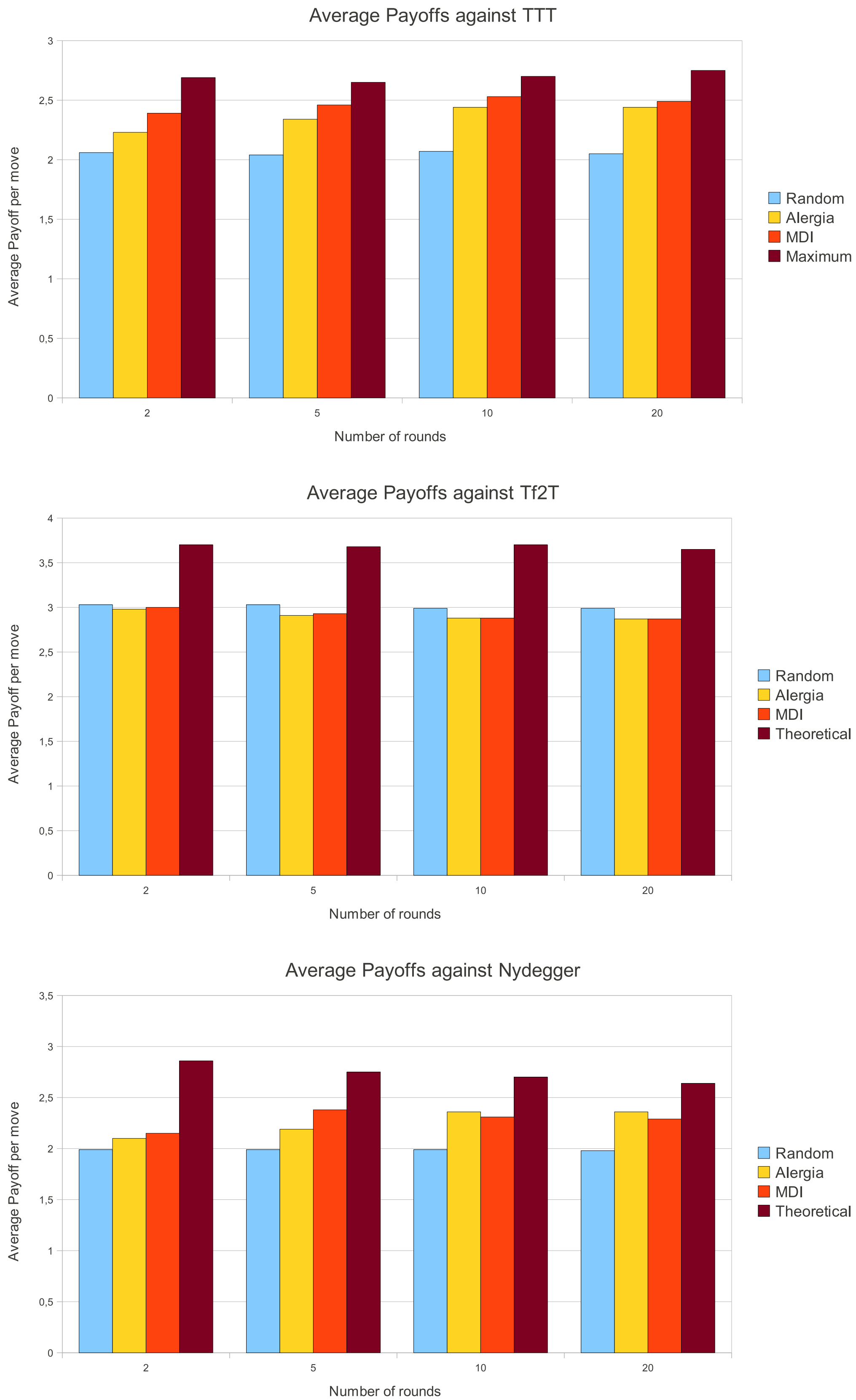

Figure 5.2: Average payoff against different opponents when we vary the number of rounds in the game

number of rounds for a fixed number of movements affects the quality of the game results.

Figure 5.2 shows that on average, MDI algorithm performs better that ALERGIA taking into account the average payoff per move. Both algorithms seems to improve with the number of rounds, till we arrive at 10 rounds per game, where the improvement stabilizes. One remarkable thing is that when competing against the Tf2T opponent, the random player performs better than whichever of our algorithms. This is because of the structure of the opponent and the chosen payoffs for the Prisoner's dilemma game. The opponent reacts to 2 consecutive defections with the defect action, but it returns to the cooperative setting whenever a cooperation happens. Given the payoffs the most profitable strategy could be to alternate defections and cooperations on each single state, but our agent can't learn this strategy because it can not query which moves it issued (that would break our MDP hypothesis also).

5.2.2 Match length

Here we study how the match length affects our game results in the competition. Remember that match length relates to the length of the words the agent is learning and in turn to the complexity of the prefix tree that the algorithm must manipulate.

Again the game consists on conducting 500 movements and fixing the number of rounds to 5. Results can be seen in Figure 5.3. This time the result are a little bit different than when varying the number of rounds. The differences between the ALERGIA and MDI algorithm seems lesser when only varying the match length. Even from the results it could be inferred that the MDI algorithm is a little bit worse than its competitor. On the other hand, we can see that changing the match length does not solve the structural problem with the Tf2T opponent.

5.2.3 Alpha

For this experiment we study the effect of the α parameter on each of the algorithms. Remember from Sections 3.4.3 and 3.4.4 that the α parameter controls how liberally could we consider the merge of different states. Lower alphas tend two create automata with less states while increasing the value allows the addition of more states in the resulting automaton.

Results can be seen in Figure 5.4. It can be seen that when varying α, best results are obtained both for low and high values of the parameter. Automata with a low number of states are somehow "averaging" the symbol frequencies because there are few states that can introduce discontinuities in the rate of appearance. On the other hand automata with a high number of states tend to mimic better the structure of the opponent and thus offer better results. So there is a middle range of α values that begin to create structure in

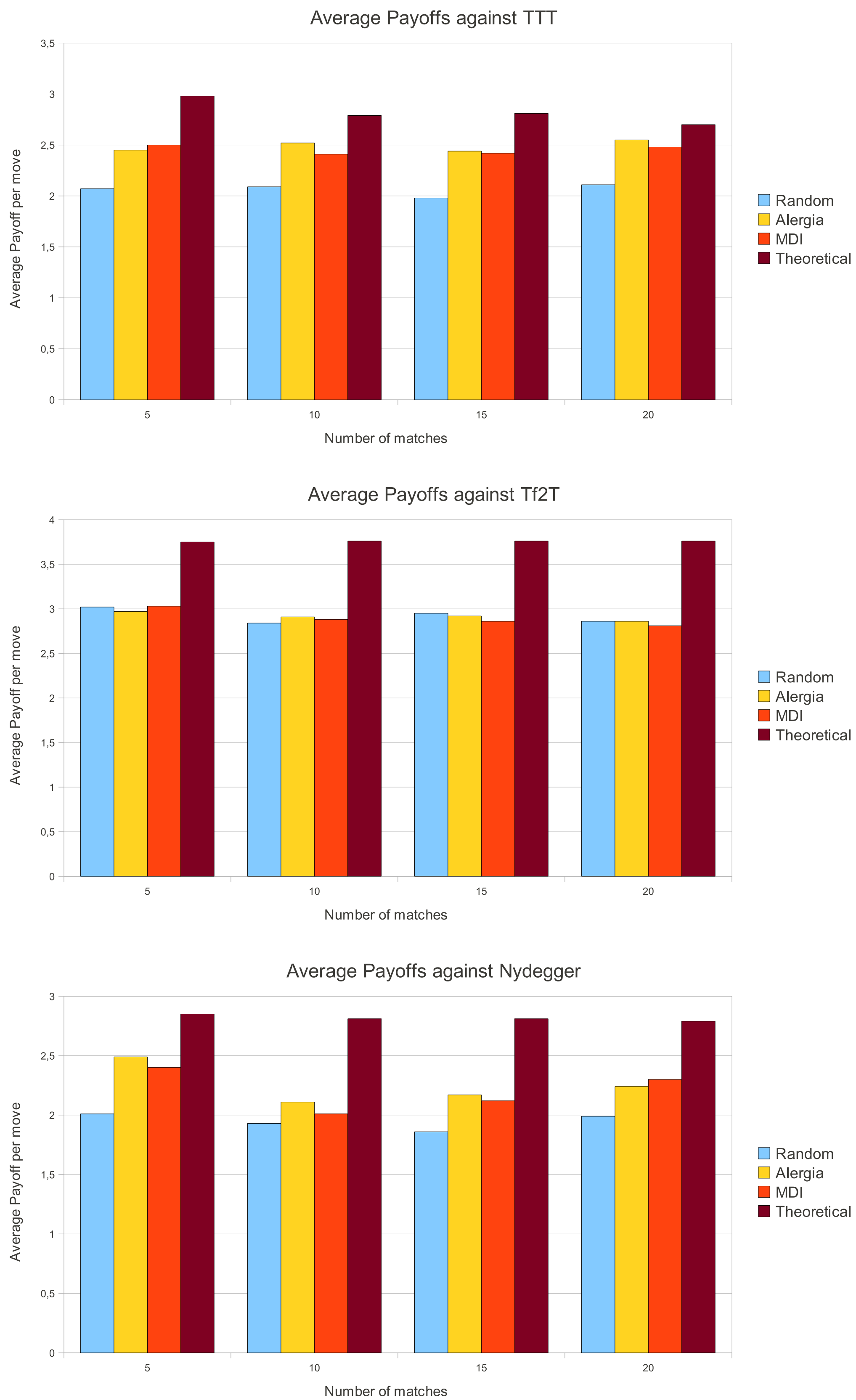

Figure 5.3: Average payoff against different opponents when we vary the match length within each round

the automata but with diverging results from the ones truly representing the opponent.

5.3 RoShamBo

For the automata in Section 5.2 we had some examples of real automata with a meaningful behaviour in the competition. In this section we are going to study how the methodology compares when automata are created randomly. To that end we have created 4 different automata with a random procedure that picks at random both the structure (the transitions) and the emitting symbol probabilities. The chosen game for these experiments will be the RoShamBo (see sec. 2.3.1) that introduces three symbols to pick from and a zero-sum structure of the rewards. In Appendix B is represented the structure of the automata. The emitting probabilities have been kept apart in order to improve legibility.

5.3.1 The experiment

The experiment consists on creating a game composed of 10 rounds with 20 matches per round and 10 moves per match, adding up to 2000 moves per game. Figure 5.3.1 show the results of our learning agent when confronting to the 2-state and 5-state RoShamBo opponents. For the 2-state opponent, both learning algorithms score below the random opponent (depicted as the inverted triangles in the figure) and when the alpha parameter goes beyond a threshold the learned automata acquires enough complexity and performs almost optimally. For the 5-state automaton, both algorithms can not reach near optimal performance despite whichever α we use for training. However the performance is always better than choosing randomly actions, so some kind of learning has taken place in the learning environment.

Figure 5.3.1 shows the results against the 10-state and 20-state RoShamBo opponents. Again our learning agents behave better than the random opponent with the only noticing particularity that in the 10-state games, the MDI algorithm performs near optimally for low alpha values, then the performance drops below the random level for middle values, and regains for higher values.

Another interesting property is that the experiments present a property worth mentioning. The complexity of the learned automata scales slowly in the case of the MDI agent and grows rapidly in the case of the ALERGIA algorithm.

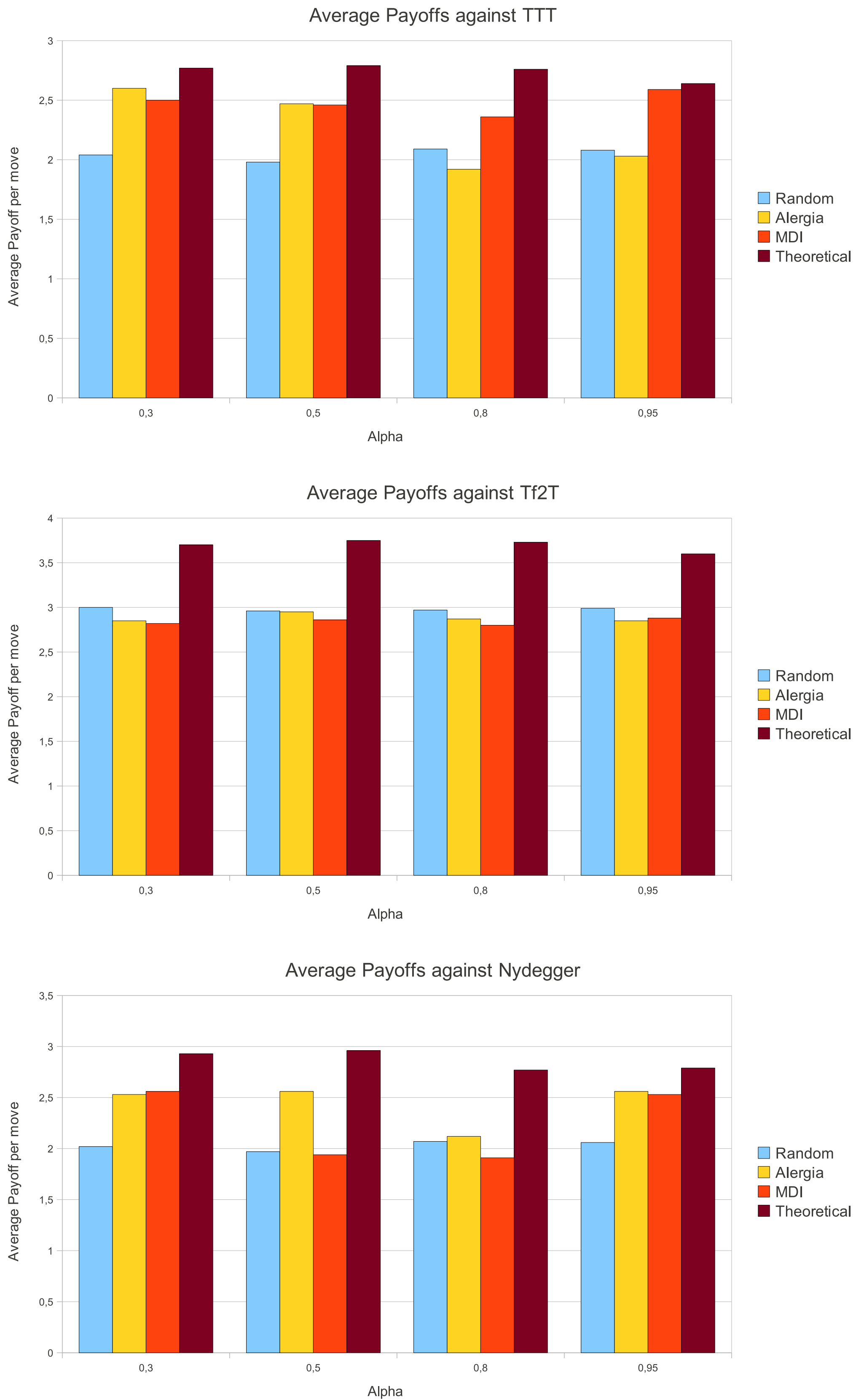

Figure 5.4: Average payoff against different opponents when we vary the alpha parameter

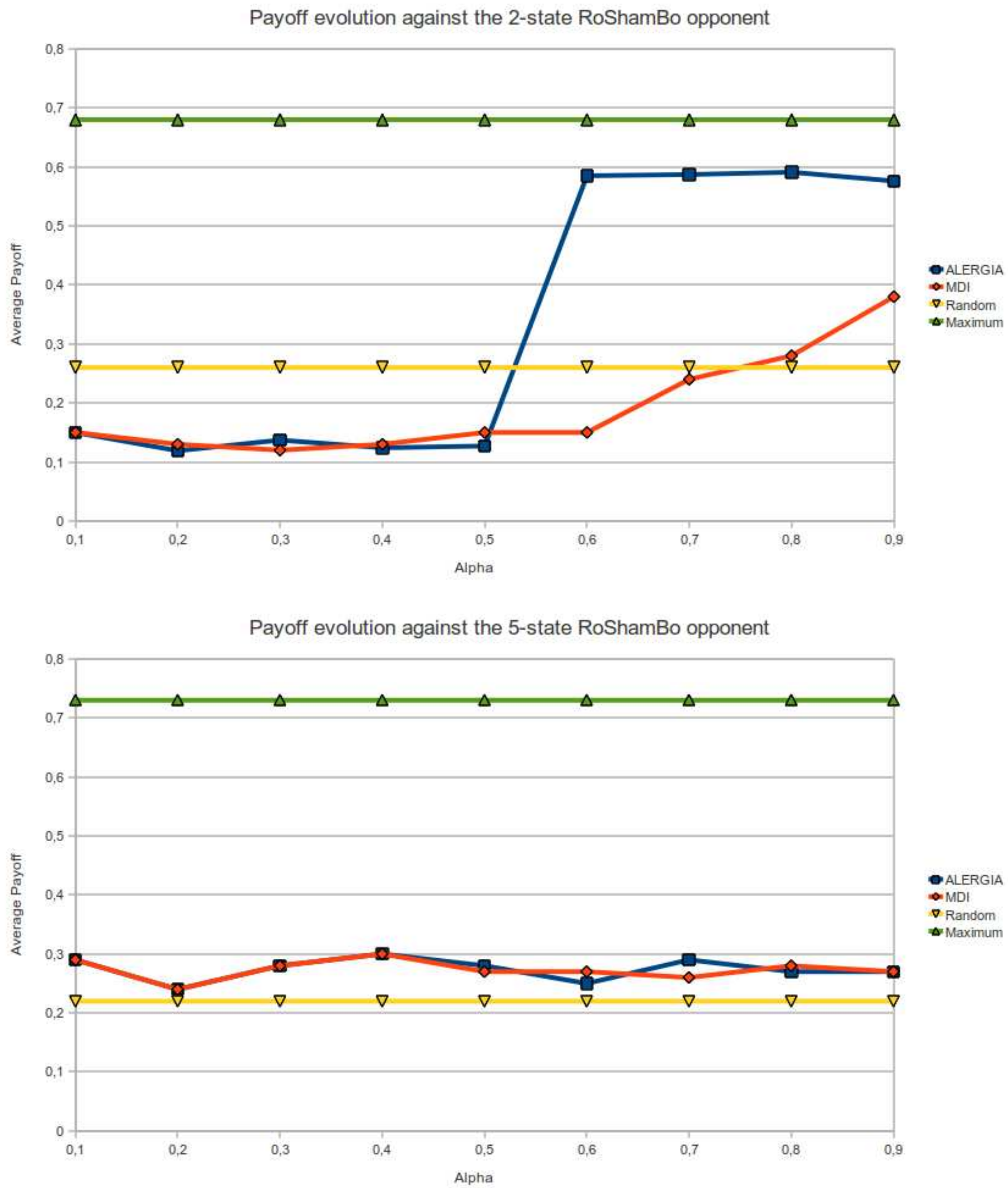

Figure 5.5: Results against the 2-state and 5-state RoShamBo opponents

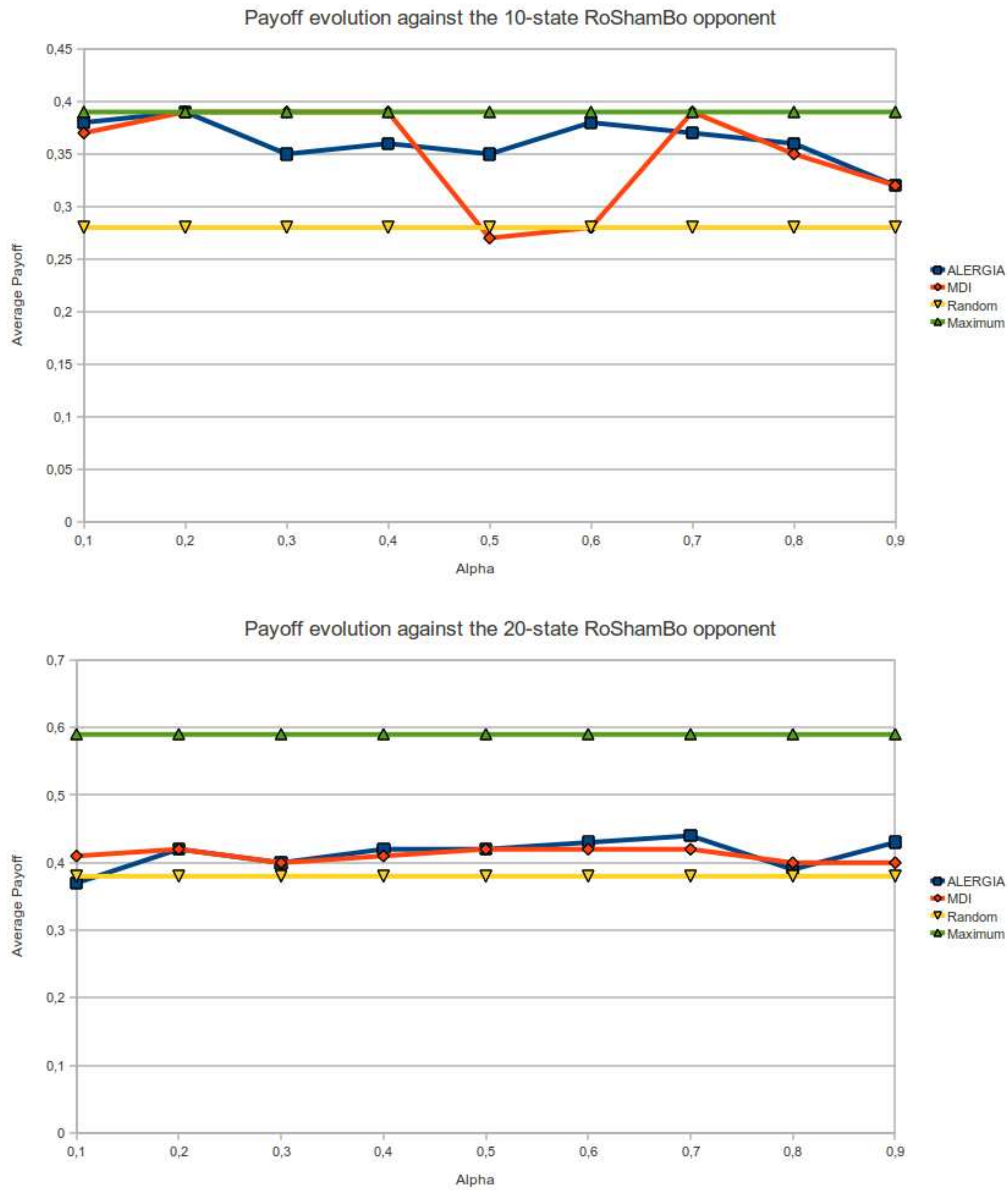

Figure 5.6: Results against the 10-state and 20-state RoShamBo opponents

Chapter 6

Conclusions

In this Master's Thesis we have explored a novel technique to learn agent behaviour in a competitive environment where Reinforcement Learning procedures are used routinely. The main tool employed in that task has been the algorithms for learning probabilistic finite state automata ALERGIA and MDI.

The range of opponents that our agents could learn were constrained to the class of Stochastic Moore Machines, finite automata that are deterministic in their transitions but that have probabilities associated to the symbols that they emit when the automata lands in one of the states.

However, the algorithms before mentioned learn PDFA not transducers like the Moore Machine. In this thesis we have used the GIATI algorithm framework and extended its usage beyond the N-gram techniques where it was initially tested. With this algorithm we can perform translations forward and backward between an extended language and the pairs of input-output symbols characteristic of the transducers.

Once we had our working hypothesis about the Moore Machine that our opponent holds, we must devise the best strategy to outperform that policy. In order to accomplish this task we presented the necessary operations to convert from a Mealy Machine to a Markov Decision Process. When the conversion completes, we use well known algorithms to devise the best strategy for each state.

In order to test our ideas, we developed a game infrastructure where different players could interact with each other and gathered results on the games played. Results were very promising since we obtained consistently better results than the random strategy. The random strategy is the best strategy in these games when the player is uninformed about the opponent strategy, so getting an advantage over the purely random move generation shows that we were in the right track.

6.1 Future work

The work done in this thesis ranged among several topics (learning PDFAs, transducers, opponent modelling, Markov Decision Processes, etc...) that could not be explored in full detail given the time constrains of a Master's Thesis. So picking one of the topics and performing a full research on them could be very insightful. In particular these are some research lines that could be followed in the future:

- **Strategy exploration**. In this thesis, when exploring the opponent strategy (that is, to walk over all the states and transitions in the unknown Moore Machine) our learning agent could only rely on the moves generated randomly in the first round of the game and on the states and transitions that could be reached with the inferred rules. This could leave some parts of the Moore Machine (i.e. the opponent's strategy) unexplored. More research on how to balance the trade off between exploration and exploitation could be tackled in order to infer more efficient opponents.

- Related to the previous point is the fact that the algorithms employed when learning the PDFA do not **take into account the payoff matrix** of each of the games. For instance, when faced with how to order states of the Prefix Tree Acceptor, the algorithms rely on the lexicographical heuristic. This is a valid heuristic but much more value could be obtained if that heuristic was related to the payoff matrix and the merging of states could take this into account.

- In the game structure that we presented in this thesis, games and matches are preset to a fixed number of moves. Although this could be reasonable in this experimental sandbox it could not make sense in real world environments. For instance, imagine that the behaviour of a user in a website is the opponent we are trying to model with our technology. Clearly the actions that this user performs, can not be constrained to a fixed size and they will vary in time. **Allowing variable match lengths** and how this affects performance is a valid research extension.

- Once the agent forms the hypothesis about the opposing agent, it runs freely and does not take into account the symbols that the agent is spitting in response to their actions until the next learning round. A very promising venue of research could be to extend the rules governing the inferred automaton in order to take into account the symbols emitted by the opposing agent. This could lower the uncertainty that we have about in which state the Moore Machine currently is and take advantage of it. Extend this thesis using techniques employed learning **Partially Observable Markov Decision Processes** .

- Compare the results of this thesis with the results obtained using Reinforcement Learning techniques

- Extending whichever of the decisions that were taken when deciding the course of the thesis. For instance, how other algorithms for learning PDFAs behave in this setup, other policies and parameters for learning the Markov Decision Processes, extending the range of models that could represent an opponent beyond the Moore Machine, etc…

Appendix A

JSON format for a game

Games are specified using the JSON data [Cro]

```
{
    "exp1":{
        "player1":{
            "type":"ALERGIA",
            "alpha":0.7
        },
        "player2":{
            "type":"MOORE_MACHINE",
            "filename":"../resources/prisonerMooreMachine.automata"
        },
        "game":{
            "type":"PRISONER",
            "numRounds":2,
            "numMatchesRound":5,
            "matchLength":5
        }
    }
}
```

Appendix B

Automata for the RoShamBo experiments

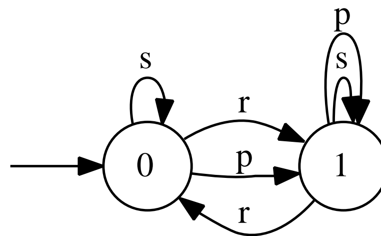

Figure B.1: 2-state automata employed in the RoShamBo Experiments

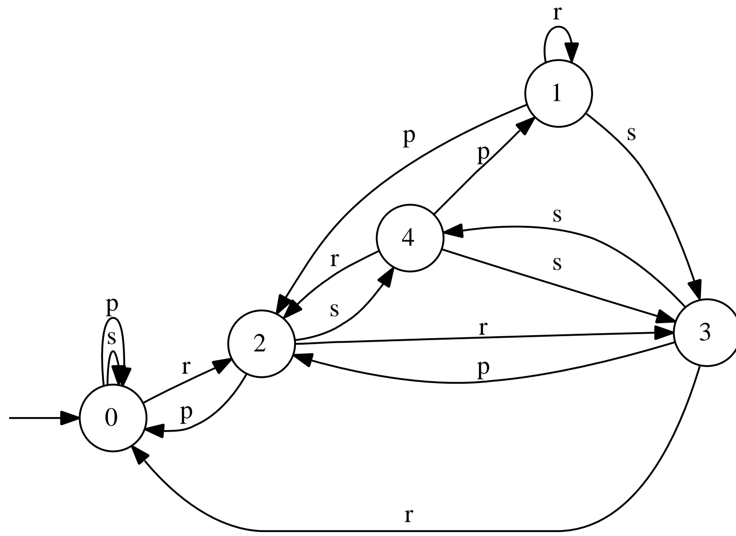

Figure B.2: 5-state automata employed in the RoShamBo Experiments

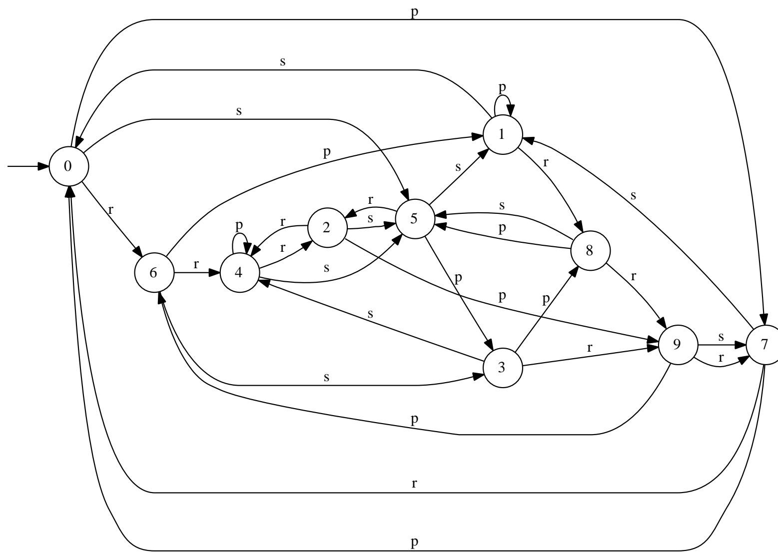

Figure B.3: 10-state automata employed in the RoShamBo Experiments

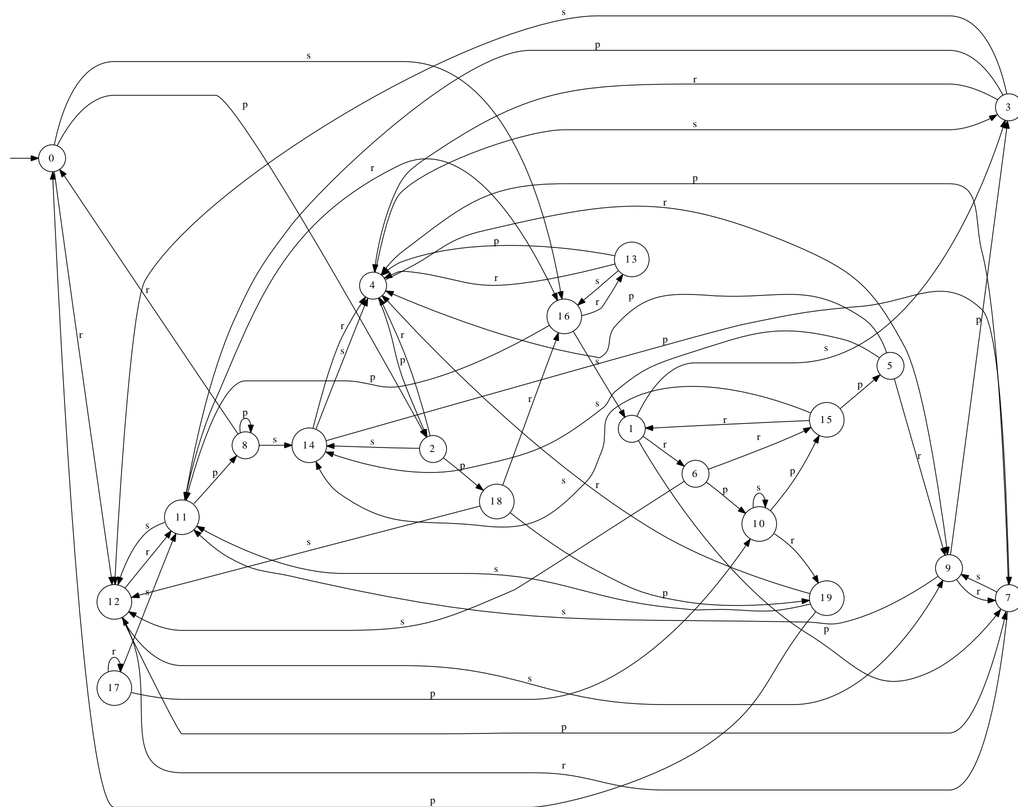

Figure B.4: 20-state automata employed in the RoShamBo Experiments

Bibliography

[ASMO97] Manuel Alfonseca, Justo Sancho, and Miguel Martínez-Orga. *Teoría de lenguajes, gramáticas y autómatas*. R.A.E.C., 1997.

[Axe84] Robert Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.

[Ber95] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[Ber09] Jean Berstel. *Transduction and Context-Free Languages*. Teubner-Verlag, 2009.

[BT96] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*, volume 3 of *Optimization and Neural Computation Series*. Athena Scientific, 1996.

[CM96a] David Carmel and Shaul Markovitch. Learning models of intelligent agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 62–67. AAAI Press, 1996.

[CM96b] David Carmel and Shaul Markovitch. Opponent modeling in multi-agent systems. In Gerhard Weiss and Sandip Sen, editors, *Adaption And Learning In Multi-Agent Systems*, volume 1042 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1996.

[CM98] David Carmel and Shaul Markovitch. Model-based learning of interaction strategies in multi-agent systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):309–332, 1998.

[CM99] David Carmel and Shaul Markovitch. Exploration strategies for model-based learning in multiagent systems. *Autonomous Agents and Multi-agent Systems*, 2(2):141–172, 1999.

[CO94] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In Rafael C. Carrasco and Jose Oncina, editors, *Proceedings of the Second International Colloqium on Grammatical Inference and Applications (ICGI94)*, volume 862 of *Lecture Notes in Artificial Intelligence*, pages 139–152, Berlin, September 1994. Springer Verlag.

[Cro] Douglas Crockford. Json specification website. `http://www.json.org`.

[CV04] Francisco Casacuberta and Enrique Vidal. Machine translation with inferred stochastic finite-state transducers. *Comput. Linguist.*, 30(2):205–225, 2004.

[CVP05] Casacuberta, Vidal, and Pico. Inference of finite-state transducers from regular languages. *PATREC: Pattern Recognition, Pergamon Press*, 38, 2005.

[Hay08] Simon Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 3rd edition edition, 2008.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison Wesley, 2001.

[LR57] R. Duncan Luce and Howard Raiffa. *Games and Decisions: Introduction and Critical Survey*. Dover Publications, 1957.

[Onc92] P. Oncina, J.; García. *Identifying regular languages in polynomial time*, chapter -. World Scientific Publishing, 1992.

[Put94] Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[PW04] Lin Padgham and Michael Winikoff. *Developing Agent Systems: A Practical Guide*. Series in Agent Technology. Wiley, 2004.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[Skl99] David Sklansky. *The Theory of Poker*. Two plus Two Publishing, 1999.

[Tho00] Franck Thollard Thollard. Probabilistic dfa inference using kullback-leibler divergence and minimality. In *In Seventeenth International Conference on Machine Learning*, pages 975–982. Morgan Kauffman, 2000.

[Woo02] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 1st edition, June 2002.