Reinforcement Learning on the Combinatorial Game of Nim

ERIK JÄRLEBERG

KTH Computer Science and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2011

Reinforcement Learning on the Combinatorial Game of Nim

E R I K J Ä R L E B E R G

Bachelor's Thesis in Computer Science (15 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2011
Supervisor at CSC was Lars Kjelldahl
Examiner was Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2011/
jarleberg_erik_K11054.pdf

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

This paper will investigate the implementation of the Q-learning reinforcement algorithm on an impartial, combinatorial game known as Nim. In our analysis of impartial games and Nim, an already established optimal strategy for playing Nim will be presented. This strategy will then be used as an exact benchmark for the evaluation of the learning process.

It is shown that the Q-learning algorithm does indeed converge to the optimal strategy under certain assumptions. A parameter analysis of the algorithm is also undertaken and finally the implications of the results are discussed. It is asserted that it is highly likely that the Q-learning algorithm can be effective in learning the optimal strategy for any impartial game.

Contents

Chapter 1

Introduction

The report at hand is part of the course "DD143X Degree Project in Computer Science" which is given by the School of Computer Science and Communication at the Royal Institute of Technology in Stockholm, Sweden.

In this report we shall examine the mathematical theory of the combinatorial game of Nim, as well as an implementation of the Q-learning algorithm to the same game.

1.1 Background

The game of Nim is a two-player mathematical game of strategy that has been played for centuries. It is said to originate from China but its origins are largely unknown.[1] The earliest European references are from the 16th century. It was given its current name in 1901 by Charles L. Bouton of Harvard University in a paper where he describes a complete mathematical theory of the game.[2]

The game is played between two players where each player takes turn in picking items from three heaps. You may pick any number of items, but you must pick atleast one and all items must be from the same heap. The player which picks the last item(s) win.

Below is an an example of how a typical game of Nim might look:

The three heaps have 5, 7 and 10 items. Alice and Bob are playing.

Alice begins by removing 4 elements from set 1. The resulting state is 1, 7, 10.
Bob removes 7 elements from set 2. The resulting state is 1, 0, 10.
Alice removes 9 elements from set 3. The resulting state is 1, 0, 1.
Bob removes 1 element from set 1. The resulting state is 0, 0, 1.
Alice removes 1 element from set 3. The resulting state is 0, 0, 0 and Alice wins.

Alice won since she removed the last element from set 3.

In general, Nim can be played with any number of heaps, but in this thesis we will assume that it is played with only three distinct heaps.

Nim has also been shown to be isomorphic to all other impartial games. What this means – rather simplified – is that all games of a certain type (impartial games) in some sense bear the same structure as Nim[8]. That is, at a certain level of abstraction, they are in a sense equivalent. This makes the game even more interesting from a mathematical analysis point of view. Impartial games will be defined in a later chapter.

1.1.1 Machine learning and reinforcement learning

A more detailed account of the concept of reinforcement learning will be given in Chapter 3. For now a mere overview of the more general field of machine learning will be given.

Machine learning is the branch of artificial intelligence which concerns itself with the algorithms and methods used for allowing computers to modify its behavior based on empirical data[13]. This is a very broad definition indeed and as such, machine learning has attracted researchers from a vast array of different scientific disciplines such as neuroscience, cognitive science, mathematics and of course, computer science among others. Learning algorithms are often applied whenever we need to develop a program that solves some problem given empirical data. For example, the task of facial recognition is something which we humans can perform very easily, yet we cannot easily give a clear definition of the underlying mechanism. This fact makes it very hard to program a computer to perform the task of facial recognition without appealing to the field of machine learning.

It is also necessary to utilize learning algorithms when we are facing a task for which the programmer cannot possibly predict the exact nature of the problem (often since it may be dependent on contingent circumstances not known at compile time). For example, this could be a robot developed by NASA to explore some planet and due to the distance it might not be viable for remote control. The robot then must achieve some sort of autonomous control and be able to perform its tasks even though the robot hasn't been explicitly programmed what to do in the unknown environment. Machine learning is used actively in the financial market as well, in trying to discover patterns involving customer behaviors. Speech recognition and handwriting recognition are also areas in which machine learning algorithms excel.

In this thesis, we will focus entirely on an algorithm called the Q-learning algorithm and which is part of what is called reinforcement learning. This will be explained more thoroughly in later chapters. In short, reinforcement learning means that the program (or agent) learns by being rewarded or punished based on its actions. This causes the agent to adapt based on this feedback from its environment. Described on this abstraction level, it bears many similarities to classical conditioning occurring among most biological creatures.

1.2 Purpose

Nim is an intriguing game in several ways. For one, it has a complete mathematical theory accompanying it and the theory is surprisingly simple. This is fascinating in its own right, since most games do not have such a theory.

The mathematical theory of Nim — in essence — describes an optimal way of playing which completely determines the outcome of the game given the initial game configuration. This is under the assumption that both players play in an optimal manner. On the other hand, if one of the players does not play in an optimal manner, then it can be shown that the optimal player will win no matter the starting position.

These two properties of Nim, namely the simplicity of its theory as well as the existence of an optimal strategy, was the reason for choosing it as the game to which the Q-learning algorithm was applied.

The existence of an optimal strategy provides us with an exact benchmark for measuring the performance of a learning agent. This is used to measure the effect of changing the different parameters of the Q-learning algorithm.

Also, the optimal agent is used as a "master" to which we will train our learning agents. One of the main points of this thesis is to examine the importance of the training partner for the learning agent. However, the main problem statement of the thesis is this: Will the Q-learning algorithm converge to the established optimal strategy for the game of Nim?

If it does, under what assumptions? It might learn the optimal strategy when playing against an optimal agent, but can it discover the theory behind Nim on its own? If so, the Q-learning algorithm might prove to be a valuable tool in the analysis of strategies for games in general.

Part I

Theory

Chapter 2

Combinatorial games

2.1 Overview

This chapter presents a brief and concise overview of the basic terminology of combinatorial games in general and the properties of impartial games in particular.

We will refrain from giving the complete formal definition of impartial games (the formal definition of extensive-form games as defined in game theory) as it is not necessary for our purposes.

2.2 Basic terminology and properties of impartial games

Games have *positions* (or *states* as we will refer to them later on) in which *players* can perform *actions* leading to new states. The set of all possible states in a game will be denoted by $\mathcal{S}$.

A player of the game may perform an *action* which may change the current state into a new state according to some *transition model*. A *sequential game* is a game in which a player chooses an action before the other player(s) choose their actions. It is essential that the later players have some information regarding the moves performed by the earlier players. Otherwise the difference in time would have no effect on the strategy employed (such games are called *simultaneous*) [3]. By a *move* we denote the act of performing an action (in some state).

Combinatorial games are a subset of sequential two-player games in which there is no element of chance and each player has *perfect information*. Perfect information means that both players have complete information regarding the actions performed by the other player. Games which involve no element of chance are called *deterministic*.

Definition 1. *An impartial game is a combinatorial game which satisfies three basic properties:*

1. *The actions available to a certain player depend solely on the state of the game and not on which player's turn it is.*

2. *The player which has no possible action to perform loses.*

3. *The game will always come to an end, or equivalently, there is no sequence of actions for which the game will continue forever.*

Nim is an impartial game. This will be shown in a more rigorous manner later on. An explanatory motivation is given below.

Since any set of elements removed by one player might just as well be removed by the other player we have that the first property is satisfied. An example of a game which does not satisfy the first property is chess. In chess, each player has ownership over certain pieces and may only perform actions on these pieces. The second property follows directly from the definition of the game and the third property is clear when we see that each action performed must remove at least one element from some set. Thus we will reach the end state in a finite number of moves.

2.3 Backward induction

In our analysis of Nim we will apply a technique called backward induction. It has been used since the advent of game theory and was first mentioned in the classic book *Theory of Games and Economic Behavior* (von Neumann, & Morgenstern, 1953)[4] which established game theory as a field in its own right. Cruical to the viability of backward induction is the finiteness of the possible state sequences.[5] We will begin with a simple example of the application of backward induction upon a very simple version of Nim (also known as a take-away game[6]).

2.3.1 Take-away game

We will define this impartial game as follows:

1. There is a set of in total 12 elements.

2. A valid move is to remove either 1, 2 or 3 elements from this set. At least one element must be removed at each turn.

3. The last player to perform a move (i.e. reducing the set to 0 elements) wins.

Backward induction works by working out which possible moves might have caused a game to reach the endstate (s).

In our example from above, suppose we have 1, 2 or 3 elements left in the set. The player whose turn it is to move will win, since he can remove all the remaining elements and consequently win. Suppose instead that we have 4 elements in the set. Then the player whose turn it is to move must place the resulting state with either 3 (remove 1), 2 (remove 2) or 1 (remove 3) element (s) left. This is illustrated in Figure 2.1. As a result, the current player will not win. Continuing in this manner, we can work our way backwards and classify all the different states into two sets.

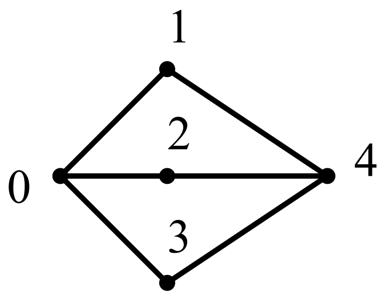

Figure 2.1. Last four states of the Take-away game

Table 2.1. State set partition of the take-away game

Position	Optimal action	$\mathcal{N}/\mathcal{P}$
0	-	$\mathcal{P}$
1	Remove 1	$\mathcal{N}$
2	Remove 2	$\mathcal{N}$
3	Remove 3	$\mathcal{N}$
4	-	$\mathcal{P}$
5	Remove 1	$\mathcal{N}$
6	Remove 2	$\mathcal{N}$
7	Remove 3	$\mathcal{N}$
8	-	$\mathcal{P}$
9	Remove 1	$\mathcal{N}$
10	Remove 2	$\mathcal{N}$
11	Remove 3	$\mathcal{N}$
12	-	$\mathcal{P}$

It can be shown that the state-space for any impartial game can be partitioned into a $\mathcal{P} - set$ and a $\mathcal{N} - set$. This will be given a recursive definition in the next section. For now, observe the division of the state space of the simple Take-away game in Table 2.1.

The intuition behind the naming of the sets is that if the current state s belongs to $\mathcal{N}$ then the *next player* to perform a move will win under the assumption that the player moves to a state belonging to the $\mathcal{P} - set$. In this way, when it is the other player's turn to move, that player will face a state which belongs to the $\mathcal{P}$-set and thus the previous player (i.e. the one that started) will eventually win.

From the above result of backward induction, we can see that the player which starts in the 12th position in the take-away game must lose given that the other player plays in an optimal manner.

2.4 The partition of the state set

The formal definition of the two sets $\mathcal{N}$ and $\mathcal{P}$ is given below. The state set of any impartial game may be split into these two sets.

It is as such a general framework for analyzing impartial games. If a correct partition is found, then we also have an optimal strategy for the game.

Definition 2. *$\mathcal{P} \subset \mathcal{S}, \mathcal{N} \subset \mathcal{S}, \mathcal{P} \cup \mathcal{N} = \mathcal{S}$ and satisfy the following inductive properties:*

1. *All terminal positions are $\mathcal{P}$-positions.*
2. *From all $\mathcal{N}$-positions, it is possible to move to a $\mathcal{P}$-position.*
3. *From all $\mathcal{P}$-positions, every move is to a $\mathcal{N}$-position.*

2.5 Nim

We will now turn our attention to the game in question of this thesis, namely Nim. We will use the theory and terminology from above in our analysis of Nim.

As we recall, a position (or state) in Nim is simply the amount of elements in the three sets.

The game is played between two players where each player takes turn in picking elements from three sets (or heaps). The initial game configuration consists of any number of elements in the three sets. Each player must remove at least one element at each turn, and it is allowed to remove any number of elements given that they all come from the same set. The player to remove the last element(s) and thus reducing all the sets to zero wins.

Definition 3. *The set of states for the game of Nim is $\mathcal{S} = \{(x_1, x_2, x_3) | x_i \in \mathbb{N}_0,\ x_i \leq n,\ i = \{1, 2, 3\}\}$ for some $n \in \mathbb{N}$.*

We will denote the inital state of the game as $s_0 \in \mathcal{S}$. The value of n above is thus $max(s_{0_1}, s_{0_2}, s_{0_3})$.

Definition 4. *The set of all possible actions for all states will be denoted by $\mathcal{A}$.*

An action in Nim will be defined as a vector with two components, namely $(\alpha, \beta) \in \mathcal{A}$. α specifies from which set (or rather which component) of the state we should remove elements and β specifies the amount.

Definition 5. *By Actions : $\mathcal{S} \to \mathcal{A}$ we will denote the set of possible actions in a given state. Actions is defined by the following map:*

$$(x_1, x_2, x_3) \mapsto \{i \times [1, x_i], i \in [1, 3]\}$$

Definition 6. *By Result* $: \mathcal{S} \times Actions(\mathcal{S}) \to \mathcal{S}$ *we will denote the resulting state when an action is performed in a specific state. Result is defined by the following map:*

$$[(x_1, x_2, x_3), (\alpha, \beta)] \mapsto (x_1, x_2, x_3) - (z_1, z_2, z_3) \;\; where \; z_\alpha = \beta, z_i = 0, i \neq \alpha$$

For example the action $(1, 3)$ would remove three elements from the first set. If we were in the state $(7, 9, 5)$ for example, we would get $Result\,[(7, 9, 5), (1, 3)] = (4, 9, 5)$

The state space is the set of all possible game sequences given an initial state.

With these definitions in place, we may show that all possible game sequences converge to the terminal state $(0, 0, 0)$.

Theorem 1.

$$\lim_{n\to\infty} s_n = (0, 0, 0)$$

Proof. All possible game sequences can be defined in a recursive manner by $s_{i+1} = Result(s_i, a)$ for some $a \in Actions(\mathcal{S}) \subset \mathcal{A}$, when s_0 is given.

From the fact that every action causes a reduction of some component of a given state, we have that $s_{i+1} < Result(s_i, a)$ (under the product order) and thus the sequence is strictly decreasing.

It is also clear that $\forall s \in \mathcal{S},\; 0 \leq s$ and thus it is bounded below and therefore all possible game sequences must converge to the unique terminal state. □

2.6 Classification of the $\mathcal{P}$ and $\mathcal{N}$ positions in Nim

We will now present the main theorem regarding Nim, namely the theorem that determines if a state belongs to the $\mathcal{P}$-positions or the $\mathcal{N}$-positions. As we noted earlier, an optimal strategy in an impartial game is to perform the actions such that the resulting state belongs to the $\mathcal{P}$-positions.

Thus if we can show which states belong to the $\mathcal{P}$-positions, we will have an optimal strategy to Nim.

Definition 7. *We define the nim-sum operator* $\oplus : \; \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *as*

$$[(x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n)] \mapsto (z_1, z_2, \ldots, z_n) \;\; where \; z_i = \begin{cases} 1 \; if \; x_i \neq y_i \\ 0 \; if \; x_i = y_i \end{cases} = x_i + y_i (mod\, 2)$$

This operator is also called the XOR (exclusive OR) operator. A state $(x_0, x_1, x_2) \in \mathcal{S}$ in can be expressed in an equivalent manner by using a binary digit representation of the components.

For the theorem to follow, assume that each component $x_i \in \{0,1\}^n, i = 1, 2, 3$ for some n, and that this is the binary representation of the state.

Theorem 2. *Bouton's Theorem*

$$(x_1, x_2, x_3) \in \mathcal{P} \Leftrightarrow \; x_1 \oplus x_2 \oplus x_3 = 0$$

In words this means that a position belongs to the $\mathcal{P}$-positions if its nim-sum is zero.

Proof. Denote by $\mathcal{P}$ the set of positions which have nim-sum zero and let $\mathcal{N}$ be the complement set (nonzero nim-sum).

$$\mathcal{P} = \{(x_1, x_2, x_3)\,|x_1 \oplus x_2 \oplus x_3 = 0\}\,,\ \mathcal{N} = \mathcal{P}^C$$

We will now show that these sets satisfy the three properties outlined in the recursive definition of the $\mathcal{P}$ and $\mathcal{N}$ set.

1. All terminal positions are $\mathcal{P}$-positions.

   The only terminal position in Nim is $(0,0,0)$ and $0 \oplus 0 \oplus 0 = 0 \to (0,0,0) \in \mathcal{P}$

2. From all $\mathcal{N}$-positions, it is possible to move to a $\mathcal{P}$-position.

   Assume $s \in \mathcal{N}$ is an arbitrary $\mathcal{N}$-position.

   $(\alpha, \beta, \gamma) \in \mathcal{N} \to \alpha \oplus \beta \oplus \gamma \neq 0 \to \exists i, \alpha_i + \beta_i + \gamma_i \in \{1, 3\}$.

   Select the highest such i (the leftmost digit). Now we have that either $\alpha_i = 1, \beta_i = 1$ and $\gamma_i = 1$ (if the sum was 3) or that one of these is equal to one (if the sum was 1).

$$\begin{array}{cccccc} \alpha_1 & \alpha_2 & \dots & \alpha_i & \dots & \alpha_m \\ \beta_1 & \beta_2 & \dots & \beta_i & \dots & \beta_m \\ \gamma_1 & \gamma_2 & \dots & \gamma_i & \dots & \gamma_m \\ \hline 0 & 0 & \dots & 1 \vee 3 & \dots & 0 \vee 1 \end{array}$$

   In either case, change the one that is 1 or choose any of the three, and change it into a 0. The resulting position will be less than the current (under the product order) since we have changed the most significant digit from 1 to 0 and it is thus a valid action.

3. From all $\mathcal{P}$-positions, every move is to a $\mathcal{N}$-position.

   Assume $(\alpha, \beta, \gamma) \in \mathcal{P}$ is an arbitrary $\mathcal{P}$-position. $(\alpha, \beta, \gamma) \in \mathcal{N} \to \alpha \oplus \beta \oplus \gamma = 0$. Since the nim-sum operator is commutative, we can without loss of generality assume that α is changed to $\alpha' < \alpha$. But then we cannot have that $\alpha \oplus \beta \oplus \gamma = 0 = \alpha' \oplus \beta \oplus \gamma$ since the cancellation law would give that $\alpha = \alpha'$ and this would yield a contradiction. This shows that the nim-sum is non-zero and thus $(\alpha', \beta, \gamma) \in \mathcal{N}$.

We have shown above that the three defining properties of the sets are satisfied and thus the theorem follows. ☐

With the theory given, we now have an optimal strategy for playing Nim. We shall conclude this chapter with an optimal algorithm for playing any impartial game given that we know the partition of the state set.

Algorithm 1 Optimal agent for an impartial game

```
Input: A percept, namely the current state $s \in \mathcal{S}$. A partition of $\mathcal{S}$ into $\mathcal{P}$-positions
and $\mathcal{N}$-positions as defined above.
  if $s \in \mathcal{N}$ then
    return $x \in Actions(s) \cap \mathcal{P}$ {Choose an action such that the resulting state is
    a $\mathcal{P}$-position.}
  else
    return $x \in Actions(s) \subset \mathcal{N}$ {All of the actions belong to the $\mathcal{N}$-positions,
    choose any.}
  end if
```

Given Bouton's theorem, we can derive an optimal agent from the above general algorithm for the game of Nim.

Chapter 3

Reinforcement learning and the Q-learning algorithm

3.1 Overview

This chapter introduces the concept of reinforcement learning and describes the main algorithm of this thesis, namely the Q-learning algorithm. Some of the terminology used in this chapter has already been introduced in the previous chapter and will not generally be reiterated.

This chapter will present the Q-learning algorithm but in a very brief manner as it is beyond the scope of this report to derive it in a complete sense. For further information, the reader is referred to any introductory textbook in machine learning or AI. The basis of this chapter is due to the book "Artificial Intelligence: A modern approach"[7].

3.2 Background

In the field of artificial intelligence, an (intelligent) *agent* is an entity that performs actions in some environment. This differs from an ordinary computer program in the sense that the agent acts autonomously and acts based on its perception of its environment. This usually entails that the agent adjusts its behavior in such a manner so as to achieve the best possible outcome.

An agent is performing the act of learning if it is is capable of improving its performance on future tasks based on its earlier experience on similar tasks[7]. Defined in this broad sense, learning is indeed a very general topic encompassing concepts from control theory, operations research, information theory, statistics etc.

The general concept of learning can be further subdivided into *reinforcement learning* and *supervised learning* (among others). Reinforcement learning can be contrasted with supervised learning. In a supervised learning situation, the agent in question is given input-output pairs and learns a function that approximates these pairs (and thus learns which outputs relate to which inputs).

Reinforcement learning however works by a different mechanism. The agent learns by being given a series of reinforcements (rewards or punishments) based on its actions. Reinforcement learning then tries to maximize the cumulative rewards received in the different states. For example in our game of Nim, the agent would be given a reward if it wins and a punishment if it loses. It is however rare that the reinforcement information is available directly after performing a certain action.

It is usually the case that the reinforcements are delayed and are available at a later time (for example when the game ends). The learning task is thus complicated by this fact and the agent must try to deduce which of its actions contributed the most to produce the final outcome of the game.

3.3 Preliminaries

In our further discussions, we will have the reader recall the functions *Actions* and *Result* as well as the set of states $\mathcal{S}$.

These functions define the environment in the sense that they define the possible state sequences (the *state space*).

We must also define a *reward function* which instructs the agent which states it should prefer (and therefore the rewards received will be crucial in deciding which actions to take). The reward for a state s is denoted by $R(s), R : \mathcal{S} \to \mathbb{R}$.

The reward function maps each state to a real number. Positive rewards are used to make some behavior more prevalent and a negative reward "punishes" the agent and makes the behavior leading to those states less prevalent overall.

A *policy* is a function which maps states to actions. That is, a policy completely determines the agent's behavior. A policy will usually be denoted by π. $\pi(s)$ thus gives the recommended action for the given state s when executing the policy π. The optimal policy is denoted by π^*.

We often speak of a state sequence which the agent perceives or acts in, this is usually denoted by $\{S_0, S_1, \ldots, S_n\}$. As an example, an agent may be in state S_i and perform an action which then puts the agent in state S_{i+1}.

There are two fundamental constans used in the Q-learning algorithm. These are the learning rate α and the discount factor γ.

3.4 Feedback

The feedback strategy used in the implementation is to reward the agent whenever it makes a move so that it reaches the goal (and thus wins the game) and punish it whenever it loses a game.

3.5 Exploration

In order to learn the Q-function for the actual environment, it is of vital importance that the agent is not too greedy. A greedy agent might choose the optimal actions, but it can still be suboptimal in the larger sense. This is because the learned model is seldom the actual true environment in which the agent belongs.

There is thus a fundamental tradeoff between *exploitation* to maximize the rewards, and *exploration* to maximize the "long-term" rewards(which will be higher the better the model is) [7]. We can only hope to achieve a good model of the environment if we have explored a reasonable amount of its states.

Definition 8. *$f(u, n)$ is called the exploration function. Any function which satisfies the following two properties may be used:*

- *Increasing in u*
- *Decreasing in n*

A simple function often used is

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

where R^+ would be an estimate of the highest possible reward and N_e is some fixed parameter. The effect of using this function would be that the agent explores each state atleast N_e times.

Since we need to classify all the states of the state space as either belonging to the $\mathcal{P}$-set or the $\mathcal{N}$-set, it is of vital importance that all states gets visited by the Q-learning algorithm.

The approach used was to choose the starting state depending on which game trial it is. That is, pick a map $\theta : [1, |\mathcal{S}|] \to \mathcal{S}$ and $\theta(n)$ would give the initial state for the n:th trial.

Usually however, one defines an exploration function used in the Q-algorithm that defines the exploration behaviour of the agent. In our case, this function is simply $f(u, n) = u$ (since the exploration is handled at a meta-level).

3.6 The Q-learning algorithm

The task then for the Q-learning algorithm is to try to achieve as good a policy as possible.

To obtain the utility function U we may solve the Bellman equation[7]:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a)U(s') \tag{3.1}$$

The Bellman equation in essence captures the very intuitive idea that the utility of a state (a measure of how "good" it is compared to other states) is the reward received in that state plus the discounted utility of the next state under the assumption that the agent chooses the optimal action.

Since the environment is deterministic in our case, $P(s'|s,a) = 1$ and this reduces to:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} U(s') \tag{3.2}$$

We will denote the Q-function by $Q(s,a), s \in \mathcal{S}, a \in \mathit{Actions}(a)$ and the value is a measure of the utility of performing the action a in the state s.

Q-values are related to the utility of a state by the following equation:

$$U(s) = \max_a Q(s,a) \tag{3.3}$$

That is, the utility of a state is simply the utility of performing the best possible action in that state.

The optimal policy is defined as

$$\pi^*(s) = \arg\max\nolimits_{a' \in \mathcal{A}} Q(s,a') \tag{3.4}$$

As such, it is simply the policy which chooses the best action possible (this is under the assumption that the Q-function is in equilibrium).

We have the following constraint equation that must be true at equilibrium when the Q-values are in accordance with the optimal policy[7]:

$$Q(s,a) = R(s) + \gamma \sum_{s'} \max_{a'} Q(s,a') \tag{3.5}$$

This equilibrium equation can be stated in words in a simpler manner. The utility of a state-action pair is the reward for that specific state as well as the sum of the utility of performing the best action in all neighboring states (since we assume that we would in each of these states execute the optimal policy). The similarity of equation (3.4) and (3.5) is no coincidence.

Finally we introduce the fundamental update equation for the Q-learning algorithm.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s,a') - Q(s,a)) \tag{3.6}$$

This update equation is calculated every time the agent performs action a in state s resulting in state s'.

With these equations in place, Algorithm 2 shows the final Q-learning algorithm.

Algorithm 2 Q-learning agent

Input: A percept, namely the current state $s' \in \mathcal{S}$ and a reward r'.
Persistent:

- Q, a table of action values indexed by state and action, initially zero
- N_{sa}, a table of frequencies for state-action pairs, initially zero
- s, a, r, the previous state, action and reward, initially zero

```
if $Terminal(s)$ then
    $Q[s, None] \leftarrow r'$
end if
if $s \neq Null$ then
    $N_{sa} \leftarrow N_{sa} + 1$
    $Q[s,a] \leftarrow Q[s,a] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
end if
$s, a, r \leftarrow s', \arg\max_{a'} f(Q[s',a'], N_{sa}[s',a']), r$
return $a$
```

We will have the reader recall that the function $f(u,n)$ above is called the *exploration* function and determines how eager the agent is to explore new states. $\alpha(N_{sa}[s,a])$ is a function allowing us to change the learning rate depending on the progression of the exploration. As seen in the algorithm, $N_{sa}[s,a]$ is simply a variable keeping track of how many times the agent has encountered a specific state-set entry. Having a function α based on this value then, makes it possible to adjust learning rates depending on the progression. For example, we may arbitrarly say that we should have a higher value of α for all the state-action pairs which haven't been visited more than 50 times, and then lowering it.

The reason for wanting to adjust the learning rate of the algorithm, is that it might be beneficial to have a faster learning rate in the initial stages of the learning process, and then successively lowering it.

The final line of the algorithm assings to s, the new state s' and it chooses the action a by choosing the action that maximizes $f(Q[s',a'], N_{sa}[s',a'])$. In our case, since the exploration function is the identity in the first dimension ($f(u,n) = u$), this reduces to $\arg\max_{a'} Q[s',a']$ which is simply the optimal policy if we've reached equilibrium. See equation (3.4).

Part II

Implementation and results

Chapter 4

Implementation

4.1 Classes

The implementation was written in C++ and compiled with the Microsoft Visual Studio 2008 compiler. It uses the C++ Standard Template Library as well as Boost[1].

Fundamentally, the program consists of three main classes called Action, State and Agent.

Regarding the representation of the state set, it is currently represented as a vector. That is, the order of the components matters. An improvement would be to interpret a state as an unordered set instead of as an ordered vector, and an action as a removal of some amount of elements from an arbitrary set containing a specified amount of elements. That is, an action would correspond to the removal of X elements from any set which has Y elements. This would reduce the number of state-action pairs, however this was not implemented but it should be noted that this would significantly improve the learning speed.

4.1.1 State

The state class uniquely determines a state $s \in \mathcal{S}$ and thus internally consists of three integers.

It provides the following public interface:

- `std::vector<Action> GetActions() const`

  This function returns a vector with all the actions which are valid from this given state. See the definition of *Actions* in Chapter 2.

- `bool IsGoal() const`

  Returns true if the state is the goal state, that is if $s = (0, 0, 0)$, false otherwise.

[1]The Boost library (http://www.boost.org/) is a collection of free peer-reviewed, open source libraries that provides extensive extra functionality to C++.

- `bool operator==(const State& B) const`

  Comparison between two state objects, returns true if the states are equal.

- `int NimSum() const`

  Calculates the nim-sum for the given state.

- `void DoAction(const Action& kAction)`

  Performs the given action on the state and thus transforms it into a new state.

- `State PeekAction(const Action& kAction) const;`

  Performs the given action on the state and and returns the new state.

4.1.2 Action

The action class specifies an action in accordance with the definition presented in the Theory chapter. Internally it consists of an integer specifying which set to remove elements from(an integer in the interval $[0, 3]$) as well as the amount.

It provides the following public interface:

- `bool operator==(const Action& b) const`

  Comparison between two action objects, returns true if the actions are equal.

4.1.3 Agent

This class in an abstract base class and it provides the framework for all the agents. The main function of the Agent-class is the behaviour function which given a state, returns an action. This is in line with the definitions given in the Theory chaper. This function is *not* defined for the Agent class, and consequently all deriving classes *must* specify the agent's behaviour. In other words, it is not possible to instantiate an agent without specifying a behaviour.

The second important function is the function which provides feedback to the agent. This function is defined for the Agent class, however it does nothing in its original form. Since it is a virtual function, any deriving class may override this definition.

- `virtual Action behaviour(const State& kCurState) = 0;`

  This function is to be implemented by any deriving class and it specifies the agents policy(strategy). Given a state, the agent recommends an action to perform in that state.

- `virtual void givefeedback(const State& kCurState, double dFeedback)`

  This function is to be overridden by any learning agent. It is not purely virtual however.

4.1.4 Three different agents

Three different agents have been implemented. These are the Q-learning agent, the optimal agent and a random agent. All of these agents derive from the common base class Agent defined above.

Random agent This agent simply overrides the behaviour function and calls the *GetActions*-function of the given state and select a random action to execute.

Optimal agent This is an implementation of of the Optimal agent algorithm for Nim as defined in Chapter 3.

Q-learning agent This is an implementation of the the Q-learning algorithm described in Chapter 3. Internally it uses a hash map for storing the Q-function.

Chapter 5

Results

5.1 Methodology

With the implementation in place, it is of essence to measure two different concepts. We want to know how good the learnt policy is and we want to know how quickly the Q-learning agent learns.

The reader is reminded that α is the *learning rate* and that γ is the *discount factor* of the Q-learning algorithm.

We want a measure of how effecient the Q-learning algorithm is for the game of Nim. This is of course dependent on the choice of the two parameters α and γ. It also depends on which agent the learning agent is facing. So in the analysis of the results, these factors have to be carefully considered. It is also of interest to see which are the optimal values for the α and γ parameters.

To measure how good the learnt policy is, we will utilize the fact that we have the optimal policy to our disposal and thus get an exact benchmark of how good any given policy is.

Note that it is possible to achieve an optimal policy even though the Q-function estimate is inaccurate [7]. So even if a policy is optimal, the Q-function might differ from the theoretical Q-function of Nim.

Definition 9. *By $U^\pi(s), s \in \mathcal{S}$, we denote the utility received when executing policy π and starting in state s.*

$$S_0 = s, U^\pi(s) = \sum_{i=1}^{\infty} \gamma^i R(S_i)$$

Definition 10. *Let U^π be the vector of utilities for the different states in the state set $\mathcal{S}$ when executing policy π.*

We define the policy loss of the policy π as $||U^\pi - U^*||$, where the norm is the L_1-norm.

The difference measures the total difference of utility when it's quantified over all possible states in $\mathcal{S}$. In other words, it measures the utility difference of executing

π instead of the optimal policy π^*.

It is as such a measure of how good or bad a given policy is, since π^* is already known.

However, a different approach was chosen. Instead of measuring the utility difference, we measure the number of states in which the agent does not perform the "correct" action. Since there is only one correct move in the states that belongs to $\mathcal{N}$, and since all moves are correct if the state belong to $\mathcal{P}$, this implies that we can count the number of states belonging to $\mathcal{N}$ in which the agent executes a wrong move.

This approach was chosen since as we noted above, the policy may be optimal even though the policy loss is non-zero. Zero deviations from the optimal policy however obviously implies policy equivalency.

We have now explained the methodology of measuring both the accuracy of the policy at a given moment as well as the learning speed. In the following sections we turn our attention to three different testing scenarios and the results.

Algorithm 3 Optimality deviations

```
Input: $\pi^i$ which is the current policy.
  $n = 0$
  $States = \mathcal{S}$
  for $s \in States$ do
    if $s \in \mathcal{N}$ then
      if $Result(s, \pi^i(s)) \in \mathcal{N}$ then
        $n \leftarrow n + 1$
      end if
    end if
    $States \leftarrow States \setminus \{s\}$
  end for
  return n
```

In practice, the only states which we have to examine are those which belong to the $\mathcal{N}$-set, since any move from a state in $\mathcal{P}$ is guaranteed to be in accordance with the optimal policy.

The optimality deviations algorithm(Algorithm 3) is called every 2500th trial and statistics are collected. 2500 was an arbitrary number chosen so as to get a reasonable amount of data for the plots.

By executing the above pseudo-code, we get an exact measure of how the Q-learning agent is progressing along towards the optimal algorithm.

The state space was limited for these tests by imposing the condition that each heap may not exceed seven items. That is, $\mathcal{S} \subset B_7(0)$, where $B_7(0)$ is the ball of radius seven of position 0 in $\mathbb{N}_0^3$.

This gives $7 * 7 * 7 = 343$ different states and in total 3087 different entries for the Q-function(state-action pairs).

5.2 Scenarios

5.2.1 Overview

The following different scenarios were tested:

1. Letting the Q-learning agent play against the optimal agent.
2. Letting the Q-learning agent play against another Q-learning agent for different values of α and γ. This is the most interesting scenario.
3. Letting the Q-learning agent play against a random agent for a fixed value of α.

We will begin the examination by examining how the discount factor value effects the learning process. This will be only for scenario two as it turns out that discount factor values other than 1 tend to give bad results. Therefore, in the subsequent sections where we examine the three different scenarios described above, we will only be concerned with manipulating the α value and having γ set to 1.

5.2.2 Discount factor

It is reasonable to assume that the optimal value for the γ parameter is 1 since the only rewards ever received are at the end states, and we should propagate this information "backwards" from the end states, to all earlier states and thus acheive our partition of the state set.

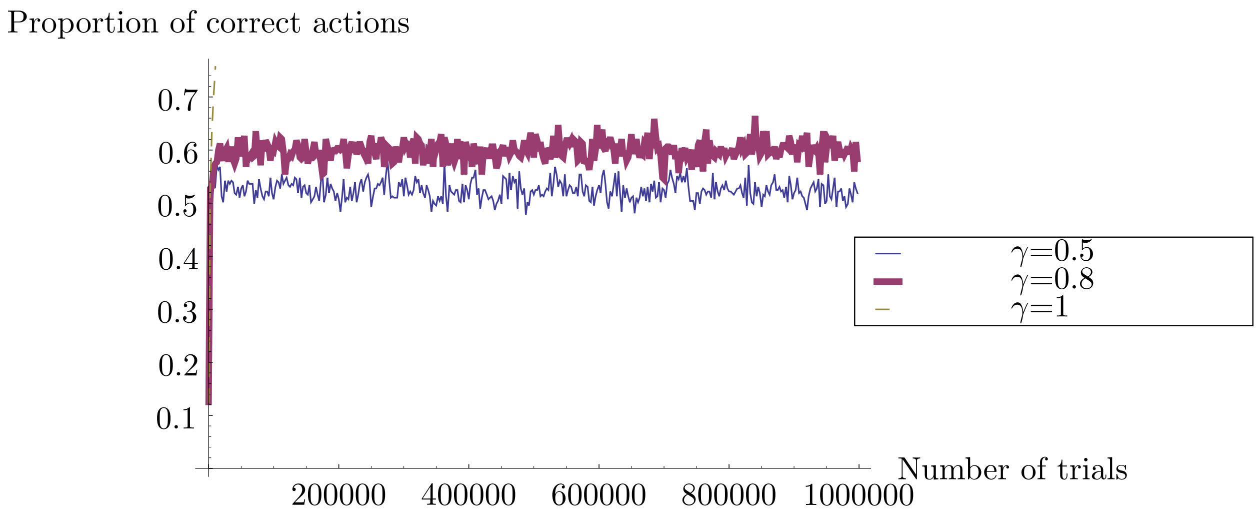

Figure 5.1. Q-agent vs Q-agent agent, $\alpha = 0.45$

This assumption was examined for three different values of γ as can be seen in Figure 5.1. It is clear that $\gamma = 1$ is the optimal parameter of those examined and it is not likely that any other discount factor (other than those examined) will be better than $\gamma = 1$. Since the $\gamma = 1$ curve converge to quickly to 1, the plot is zoomed in order for it to be visible at all (since it would be a mere straight line otherwise). In the further sections, this specific value will be given a more thorough treatment.

It is also clear that we do not acheive the optimal policy when $\gamma = 0.5$ and $\gamma = 0.8$.

5.2.3 Scenario 1: Q-agent vs Optimal agent

Since the learning agent plays against the optimal agent in this scenario, we would expect that it does indeed learn the optimal policy after some time. This is because any wrong move on the Q-agents parts *will* result in it losing the game and thus receiving negative reinforcement. Conversely, only those games in which the learning agent performs strictly correct moves will result in positive reinforcement.

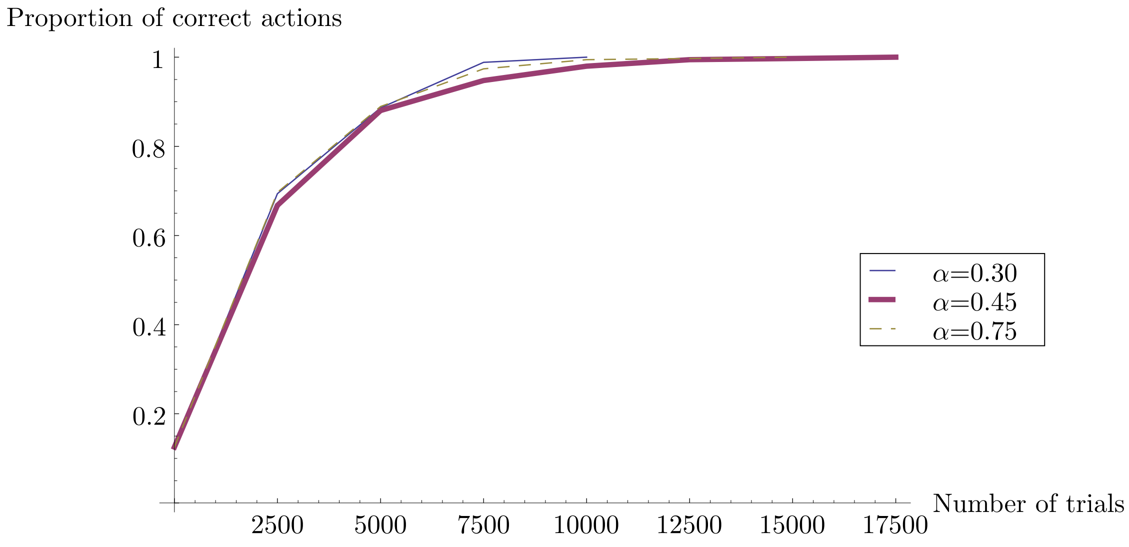

Figure 5.2. Q-agent vs Optimal agent

As seen in Figure 5.2, the Q-agent does indeed learn the optimal policy after some approximately 17500 iteration for $\alpha = 0.45$ and it learns it even more quickly using $\alpha = 0.3$.

We can conclude that this is indeed quite fast and it only takes a couple of seconds of execution time on most modern computers. It is however of little value since in practice, we cannot expect to have an optimal agent available to learn from since that would remove the whole purpose of having an agent learn a policy to begin with.

This scenario however is of importance when comparing the learning rates of the other scenarios. It will be seen later that this is undoubtedly the fastest scenario of those examined.

5.2.4 Scenario 2: Q-agent vs Q-agent

In this scenario we have two Q-agents playing against each other for several different values of alpha. The results here presented are very interesting since as it turns out – for most values of alpha – the Q-learning agent is indeed capable of autonomously learning the optimal strategy for the game of Nim for the restricted state space.

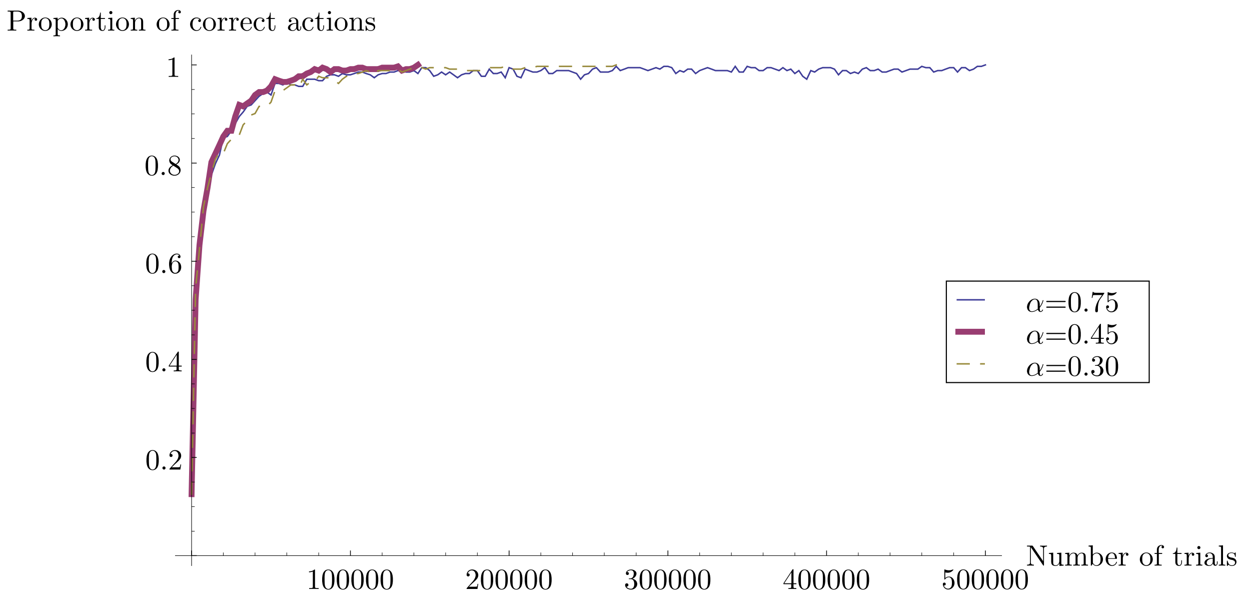

Figure 5.3. Q-agent vs Q-agent

As seen in Figure 5.3, the Q-agent does indeed converge to 100% correct actions as defined in the methodology. We also note that it seems that among the three different values of alpha in this plot, $\alpha = 0.45$ reaches the optimal policy the fastest. It reaches it in about half of the iterations required for $\alpha = 0.30$.

This can be seen more clearly in Figure 5.4, where the later phase of the learning process is emphasised. Do note that that the algorithm is construed in such a way as to abort when the optimal policy is found. In the more general case, when there is no optimal policy available to compare with, there are more general methods available to determine when computation should stop

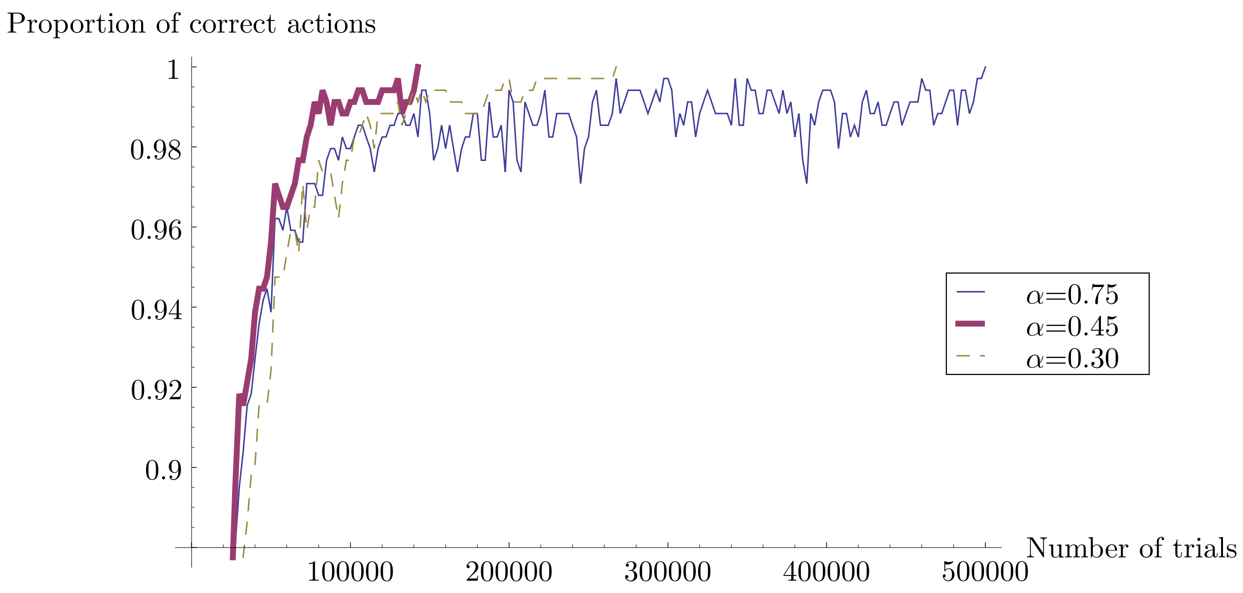

Figure 5.4. Q-agent vs Q-agent zoomed

5.2.5 Scenario 3: Q-agent vs Random agent

This scenario consisted of the Q-agent playing against an agent which chooses all its actions in a random manner. One might expect that this would not result in an agent that can play the game optimally, since it is seldom the case that a policy that is efficient against a random opponent is also efficient against a "good" player.

It turns out that in Nim, we can indeed learn the optimal policy by merely playing against a random agent. This is illustrated in Figure 5.5. However, with α set to 0.45, we did not acheive convergence to the optimal policy (it ran until the millionth iteration, however in the plot it is truncated at the 700000-th iteration).

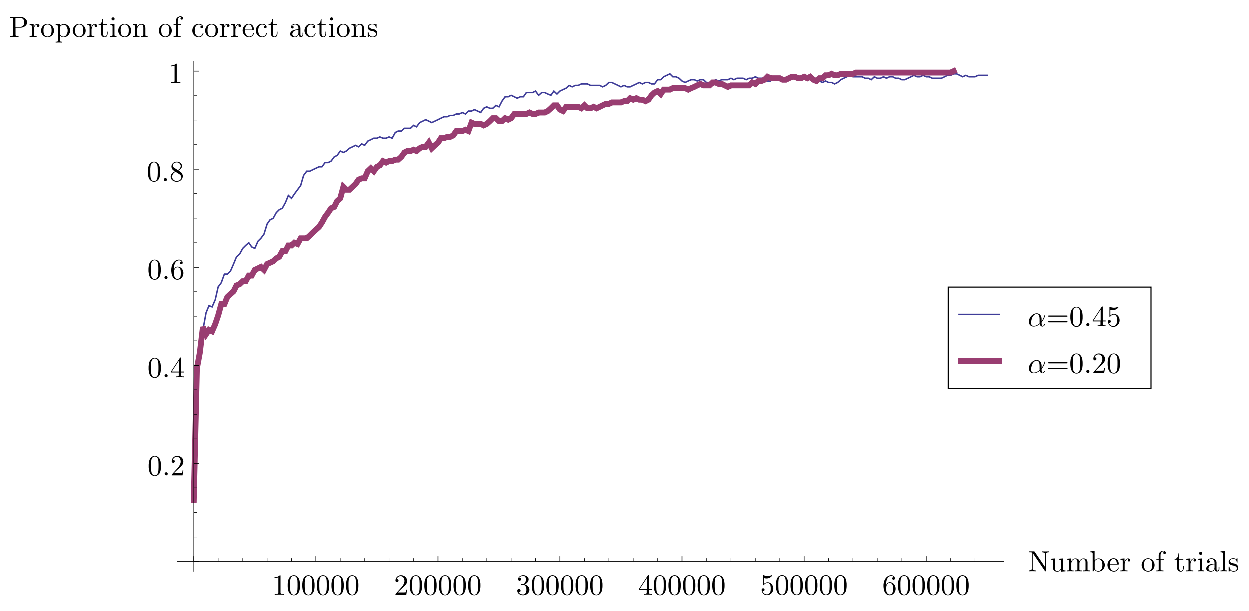

Figure 5.5. Q-agent vs Random agent

5.3 Discussion

The α-parameter regulates the learning rate of the Q-function. Some values for alpha seem to be much better than others. For example $\alpha = 0.45$ in Figure 5.4 shows that the curve for $\alpha = 0.45$ is above the curve for $\alpha = 0.75$.

With a high value of α-parameter, the changes to the Q-function are very rapid and thus it makes the graph look volatile. A higher learning rate causes the learning curve to be more volatile and often reach better policies faster, but this volatility can cause it to "miss" the optimal policy, while a lower learning rate tends to be less volatile but slower. One possible strategy to cope with this is to use a high inital α-value and lower it after some trials have been run to acheive convergence. This has not been tested.

For scenario two(Q-agent vs Q-agent), several tests were run to examine effect of the learning rate and it turns out that with $\alpha > 0.8$, we cannot expect the Q-learning algorithm to converge to the optimal policy.

On contrast, a low value of α will make the learning process take longer, but it often finds the optimal policy. In fact, all values between 0.02 and 0.8 do converge to the optimal policy for scenario two.

The number of trials in which this happens depends greatly on the initial moves of the algorithm (which are random, before the Q-function has been populated). Therefore, a plot of this has not been included as its random behaviour makes it difficult to establish any conclusions as to the exact effects of different α-values.

What can be generally concluded though, is that it fails to establish the optimal policy if the α-value is too high, and it is very slow to converge if it is too low.

Somewhere around $\alpha = 0.4$ and $\alpha = 0.5$ seems to be the optimal interval.

The lowest number of iterations was acheived at $\alpha = 0.45$ with merely 142500 iterations. This can be contrasted however with the fact that when playing against the optimal player, we learn the optimal policy with only 17500 iterations for the same learning rate. This shows that not only is the α parameter important for the rate of convergence, but the most important factor is which opponent the learning agent is facing.

This is also confirmed by examining the number of trials it took for the Q-agent to learn the optimal policy when playing against a random player. It took the learning agent 600000 iterations to converge with a learning rate value of 0.20, and it did not converge at all for $\alpha = 0.45$.

Finally, in Figure 5.6, we see the different rates of convergence when the Q-learning agent is playing against three different kinds of agents with the learning rate set to 0.45.

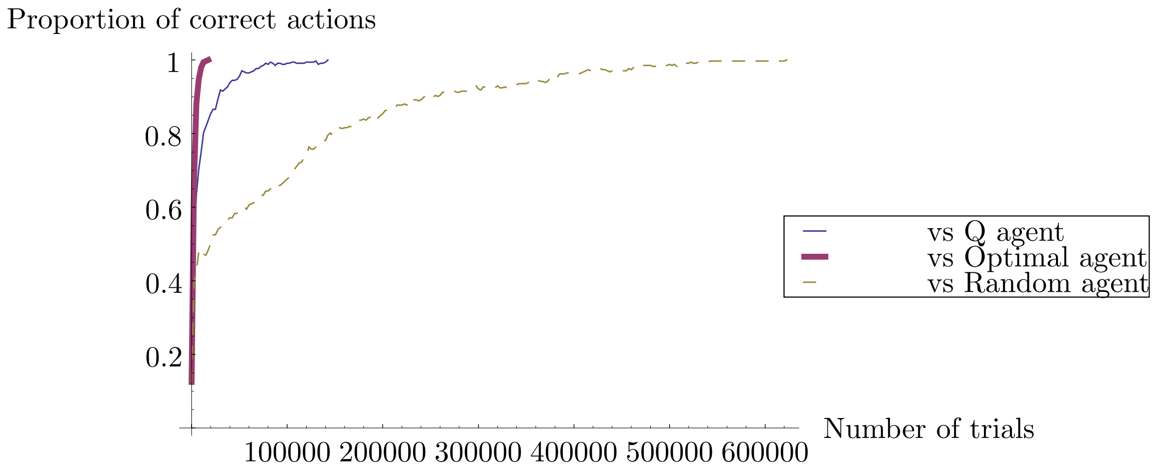

Figure 5.6. Q-agent vs three different agents($\alpha = 0.45$)

5.3.1 The resulting Q-function

When the Q-learning algorithm has learned the optimal policy, a manual examination of the resulting Q-function is undertaken. Table 5.1 shows the complete Q-function for a state set restriction of maximum two elements in each set.

The rewards have been set to 20 for winning a game and -20 for losing a game. What we notice when we examine the Q-function is that – when it has acheived the optimal policy – all positive Q-values are precisely those actions in which we go from an $\mathcal{N}$-state to a $\mathcal{P}$-state. What's more is that all the negative Q-values are precisely those states which belong to $\mathcal{P}$ and in which we are forced to take an action placing the state in $\mathcal{N}$.

As we noted in Chapter Two, all impartial games lend themselves to the possi-

Table 5.1. The resulting Q-function for all states with n=2

State	Action	Nim-sum ($\mathcal{P}/\mathcal{N}$)	Nim-sum of resulting state ($\mathcal{P}/\mathcal{N}$)	Q
$(1,1,1)$	$(2,1)$	$1(\mathcal{N})$	$0(\mathcal{P})$	20
$(0,1,1)$	$(2,1)$	$0(\mathcal{P})$	$1(\mathcal{N})$	-20
$(1,1,1)$	$(1,1)$	$1(\mathcal{N})$	$0(\mathcal{P})$	20
$(1,1,0)$	$(0,1)$	$0(\mathcal{P})$	$1(\mathcal{N})$	-20
$(1,1,1)$	$(0,1)$	$1(\mathcal{N})$	$0(\mathcal{P})$	20
$(1,0,0)$	$(0,1)$	$1(\mathcal{N})$	$0(\mathcal{P})$	20
$(1,0,1)$	$(2,1)$	$0(\mathcal{P})$	$1(\mathcal{N})$	-20
$(1,0,1)$	$(0,1)$	$0(\mathcal{P})$	$1(\mathcal{N})$	-20
$(0,0,1)$	$(2,1)$	$1(\mathcal{N})$	$0(\mathcal{P})$	20
$(0,1,1)$	$(1,1)$	$0(\mathcal{P})$	$1(\mathcal{N})$	-20
$(1,1,0)$	$(1,1)$	$0(\mathcal{P})$	$1(\mathcal{N})$	-20
$(0,1,0)$	$(1,1)$	$1(\mathcal{N})$	$0(\mathcal{P})$	20

bility of dividing the state set into $\mathcal{N}$-positions and $\mathcal{P}$-positions and we also showed a general algorithm for optimal game play in any impartial game (see Algorithm 1). Now, with this in mind, we see that we have an optimal policy if we can determine the two-set partition into $\mathcal{N}$ and $\mathcal{P}$.

The learning problem has in a sense been reduced to determining the two-set partition of the state set into $\mathcal{N}$-positions and $\mathcal{P}$-positions.

So it would be possible to apply the Q-learning algorithm to any impartial game, and when it has converged to an optimal policy, we may apply a boolean classification algorithm to determine the explicit form of the division.

If this would be done to Table 5.1 (suppose we didn't know Bouton's Theorem), the result would be exactly Bouton's Theorem dividing the state set.

The boolean classification problem would be to determine the indicator function

$$1_{\mathcal{P}} : \mathcal{S} \to \{0,1\}$$

where the state is given in binary representation (and $1_{\mathcal{P}}$ is therefore a boolean function) as defined in Chapter 2.

That is, the boolean indicator function satisfy:

$$(x_1,x_2,x_3) \in \mathcal{P} \subset \mathcal{S} \Leftrightarrow 1_{\mathcal{P}}(x_1,x_2,x_3) = 1$$

Since all boolean functions may be expressed in the disjunctive normal form (DNF)[1], this involves a search in a finite function space and is thus guaranteed to find the partition.

It is beyond the scope of this thesis to explore this further, but we can mention that a decision list learning algorithm will probably be efficient for determining the correct partition.

[1]A boolean expression is in DNF if it is a disjunction of conjunctive clauses.

Chapter 6

Conclusions

These results suggest that reinforcement learning in general – and particularly the Q-learning algorithm – might be used as a powerful method for investigating the properties of games.

What is interesting to note is that even though we've only examined the specific impartial game of Nim, the implications are wider.

If the game is impartial, the Q-learning algorithm will almost always converge to the optimal strategy under the assumption that it plays against another Q-learning player and that the parameters are reasonable. This is since the Sprague-Grundy theorem states (rather simplified) that every impartial game is isomorphic to the game of Nim. For further details, see the article "Mathematics and games" by P. M. Grundy written in 1938[8] or the book "Winning ways for your mathematical plays" by Berlekamp, Conway, & Guy from 2001[9].

These findings suggest an interesting approach to the learning of an optimal policy for impartial games in general. Assume that we are given an impartial game to which there is no known optimal strategy. The Q-learning algorithm may then be executed until we have found an optimal strategy. Two questions arise: how do we determine when we are done and what is the explicit form of the partition of the state set?

As mentioned in the Discussion part of Chapter 5, by applying a boolean classification algorithm (such as a decision list algorithm) to the Q-function, we can get a hypothesis partition of the state set.

Any candidate hypothesis can be discarded if it loses any game when starting in a position belonging to the N-positions and playing against an agent operating under the same hypothesis. This may be used as an optimality criterion.

With this optimality criterion in place, we may continuously apply it during the learning process of the Q-learning algorithm, and abort when an optimal strategy has been found. This partition of the state set is then guaranteed to be optimal.

We may also conclude that the optimal parameters for the Q-learning algorithm in the game of Nim must be a learning rate between 0.4 and 0.5 as well as a discount factor of 1. It is reasonable to assume that these values should transfer over to other impartial games as well.

6.1 The state-of-the-art

Papers on reinforcement learning are published frequently in *Machine learning* in the *Journal of Machine Learning Research*[7] and there is also the *International Conference on Machine Learning(ICML)* held annually.

It is indeed an active research area and the Journal of Machine Learning Research was in 2009 ranked on the 14th place in the article "Top journals in Computer Science" published by *Times Higher Education*[12].

A modern approach to the task of learning is that of *hierarchical reinforcement learning* methods which incorperate different levels of abstraction to the learning process.

The foundation of this approach is due to an article by J.-P. Forestier and P. Varaiya published in 1978, where they showed that behaviors at a low level of abstraction – of an arbitrary complexity – can be regarded as discrete actions by the higher-level structures invoking them[11]. It is then possible to establish a learning hierarchy. The contemporary methods that build upon this work are among others *partial programs.* A partial program defines a particular hierarchical structure for the agent, but the specific action primitives are unspecified and must be learned by reinforcement learning. These methods of reinforcement learning have proved to be very good at solving large-scale problems which are not normally viable[10] (e.g. state spaces of in total 10^{100} states).

For further reading, see the article "Machine Learning: The State of the Art" published in 2008 in *Intelligent Systems, IEEE*[14].

Bibliography

[1] Nim. *Wikipedia.*Retrieved April 4, 2011, from http://en.wikipedia.org/wiki/Nim

[2] Bouton, C L. (1901). Nim, a game with a complete mathematical theory. *The Annals of Mathematics*, 2nd Ser., 3(1/4), 35-39.

[3] Sequential games. *Wikipedia.*Retrieved April 4, 2011, from http://en.wikipedia.org/wiki/Sequential_game

[4] von Neumann, J., & Morgenstern, O. (1953). Theory of games and economic behavior. *Princeton University Press*, 3

[5] Romp, G. (1997). Game theory introduction and applications. Great Britain: Oxford University Press.

[6] Ferguson, T. S. (n.d.). Impartial combinatorial games. Retrieved from http://www.math.ucla.edu/ tom/Game_Theory/comb.pdf

[7] Russell, S., & Norving, P. (2010). *Artificial intelligence: a modern approach 3rd edition.* Pearson.

[8] Grundy, P. M. (1938). Mathematics and games. *Eureka*, 2(6-8),

[9] Berlekamp, E. R., Conway, J. H., & Guy, R. K. (2001). *Winning ways for your mathematical plays.* A K Peters.

[10] Marthi, B., Latham, S. J., & Guestrin, C. (2005). Concurrent hierarchical reinforcement learning. *International Joint Conferences on Artificial Intelligence*

[11] Forestier, J.-P., & Varaiya, P. (1978). Multilayer control of large markov chains. *IEEE Transactions on Automatic Control*, 23(2), 298-304.

[12] *Top journals in computer science.* (2009, May 14). Retrieved from http://www.timeshighereducation.co.uk/story.asp?sectioncode=26&storycode=406557

[13] Alpaydin, E. (2004). *Introduction to machine learning.* Cambridge, Massachusetts: The MIT Press.

[14] Wang, J. (2008). Machine learning: the state of the art . *Intelligent Systems, IEEE*, 23(6), 49-55.