University of Amsterdam

Thesis for graduation as Master of Artificial Intelligence

Wolf-Gradient, a multi-agent learning algorithm for normal form games

Author:
Bastiaan de Groot

Supervisor:
Joaquin Quiñonero-Candela

January 6, 2008

ABSTRACT

In this thesis a new multi agent learning (MAL) algorithm, called Wolf-gradient, is proposed. In this thesis MAL algorithms are used to learn a good strategy against opponents of which the used MAL algorithm is known. In this setting a MAL algorithm learns a good strategy by playing the same game multiple times against an implementation of the learning strategies of the expected opponents. After each round all agents update their strategy based on the strategies used by the other agents.

The performance of a MAL algorithm used in this setting depends heavily on the learning algorithms used by the other agents. To properly evaluate a MAL algorithms it would therefore be necessary to have a prior over the learning algorithms used by the opponents. This is however problematic, with the publication of a new algorithm this prior is likely to change. To postpone this problem many research focus on the intermediate goal of finding a MAL algorithm that converges to a Nash equilibrium in self-play and does not need a Nash equilibrium as input. This is seen as a minimum requirement for a MAL algorithm to be considered a good algorithm and none of the current algorithms is shown to meet this goal. Wolf-Gradient is however the first algorithm to meet this requirement.

Besides it use as MAL-learning algorithm Wolf-Gradient serves a second goal, it could be used to find Nash equilibria in N-player, M-action games. The traditional methods used for this purpose are utterly complex to implement and no implementation of these algorithms is available which can solve all the games used in this thesis. The Wolf-Gradient method on the contrary is extremely simple to implement can solve more games in practise then the traditional methods.

ACKNOWLEDGEMENTS

First of all I would like to thank my three supervisors at Microsoft Research Cambridge who guided me through the most important part of the research which resulted in this thesis: Joaquin Quiñonero-Candela, Ralf Herbrich and Thore Graepel. This thesis would not have been of the same quality without your enthusiasm, support and valuable insights.

Second I would like to thank Frans Oliehoek who helped me during the first few months to get through the massive amount of literature on this subject and in the last few months in the process of actually writing the thesis.

I would like to thank Phillip Trelford for helping me with F#, .net and getting my software to run on the cluster.

The last person to thank is Nikos Vlassis who suggested this area of research, put me in contact with the Applied gaming group of Microsoft Research and with whom I had a fruitful collaboration in Crete. The work produced during this collaboration is unfortunately not part of this thesis, but led to a better understanding of the field and was a lifetime experience, I am grateful for the opportunities you created for me.

Last I would like to thank my girlfriend, family and friends who had to miss me for such an extended period and the people I met during my periods abroad, thank you for the wonderful experience and fruitful discussions.

CONTENTS

1. INTRODUCTION

Humans have always played games. Besides that we enjoy doing it, it also helps us to develop skills which can be applied in real-life situations, for example the game of chess which has been used for centuries to teach young nobles basic strategic skills [20]. Most games are an abstraction or simplification of situation which could occur in the real world, this allows humans to concentrate only on the most important and interesting problems, games are therefore an ideal testbed for Artificial Intelligence algorithms.

1.1 What are games

In this thesis games not only refers to classical board and card games, but to any situation in which humans and/or computers interact and each player has a preference for an outcome of the interaction [31]. A game is thus played by one or more players, usually referred to as agents, and is characterized by a well defined set of rules. These rules specify: the actions each agent can take at any point in time, the information every agent has access to and the reward each agent receives in every possible game ending. This reward can be both an ordered set for example {Win, Draw,Lose} or a real number. The description of the game has to be complete and correct in the sense that all possible situations and combinations of actions by the agents lead to a new valid game state and that an agent will always prefer to take the actions in the real world that will give him the highest expected reward in the abstracted game.

1.2 Types of games

Games can differ among several dimensions, this section gives an overview of the most common classes of games, an overview of all classes discussed is given in table 1.2.

An eminent distinction is the amount of agents involved in the game. Although there are no theoretical boundaries on the number of agents, the number of agents observed in literature tends to fall in the following groups: 1, 2, N and ∞. These four groups give the analysis of the game different properties. In one player games there is no interaction with other players which simplifies the problems drastically. The two player case differs from the N player case in that there is no possibility to create coalitions between players which turns out to give a mathematical simpler solution. The games with N players are usually

Dimension	Relevant values
Number of Agents	1,2,several, infinite
Action space	Discrete/Continues
Observability	Complete/Partial
Interest	Cooperative/Competitive
Chance	Deterministic/Probabilistic

Fig. 1.1: Overview of the classification of games

considered the hardest and because of computational problems the size of N is typically small (somewhere between 3 and 5). The games with an infinite number of agents form a separate group. Their application lies mainly in the illustration of evolutionary or economical principles. Using these kind of games it is possible to analyse how large groups of agents would behave if all agents act for their own best interest. An example most readers will be familiar with is the how a producer should choose the price of a product in a free market [11].

Another common distinction made is whether the actions are discrete or continues. In discrete games an agent can chose from a finite number of actions in any point in time (for example chess). In continues games the number of actions is infinite. This is the case in for example no limit Poker: players can bet any amount of money they want (discarding the size of the chips), most research within computer science is based on games with discrete actions, infinite actions are usually discretized before the game is analyzed.

All agents in a game can have perfect or imperfect information. If all agents in a game have perfect information all agents have access to all relevant information. This is the case in for example chess. Poker on the other hand is mainly interesting because all agents have different information and have to deduce from the actions of the other players which cards they posses, this is called a game with imperfect information.

Then there is the relation between the agents. In a cooperative game the rewards received in any game ending are equal for all agents [1]. In a fully competitive game the interests of the agents are exactly opposite, this can only truly exist in a two player game, the rewards of the agents then sum up to a constant in all endstates.

The final dimension is whether chance plays a role in a game: the outcome in games like Chess and Go is determined completely by the actions of the agents and are therefore called deterministic, while in Poker and monopoly chance plays a significant role, these games are called stochastic.

[1] A more precise definition would be: If there is an affine transformation, where the multiplication is not equal to zero, for each agent such that all rewards in all endstates are equal then the game is fully cooperative

1.3 Relevance

Interactions that can be modelled as games are everywhere. In all situations where two or more agents meet, each having its own interest, their interaction can be modelled as a game. In all these situation a better understanding of how optimal behaviour can be found given the game description would be valuable in two ways: using these algorithms, computer programs can be created that play these games on our behalf and using the insights gained from these studies humans can learn to improve their performance when confronted with these games. Both examples are becoming reality. Game theory is now being taught in most economical faculties to aid the students in taking the right decisions and computer programs based on game theory are not only able to compete with world champions in several games such as Chess and two-player poker [25, 43], but are also used in real life situations to let computer programs act on behalf in real situations. For example the bidding agent on Ebay which increases your bid until it outbids all other bidders (or reaches your maximum bid) is directly based on the game theoretic analysis of auctions and is in fact an implementation of the Vickrey auction [10].

Besides the economical value, game theory also has an important scientific contribution. More and more disciplines are starting to use game theory to explain phenomena observed in nature and society. Examples are evolutionary biology [14], the evolution of language [16] and economical sciences [27].

1.4 Historic overview

Like in most fields insights gained from game theory in the last 100 years can be found back in ancient history. In the Babylonian Talmud, a compilation of ancient laws and traditions which formed the foundation for the Jewish Law, it is discusses how the complete legacy of a man with three wives should be divided after his death if his total possessions do not equal the amount promised to all his wives in his will. If the mans will states that his three wives should each receive respectively 100, 200 and 300 after his death and he leaves only 100 the Talmud states that the money should be split equal, if he leaves 300 it should be split proportional to amounts specified in his will and in case the man leaves a total sum of 200 the money should be split 50, 75, 75. This advice might seems contradictionary and has baffled Talmud scholars for years, but using recent results from cooperative game theory these solutions are shown to be in the Nucleus and are therefore considered fair[2] of this game[1].

The first game theoretic analysis of a game is from 1713. In a letter to Pierre-Remond de Montmort, James Waldegrave gives an analysis of the game le Her which would now be considered to be optimal play, Waldegrave however limited his analysis to this single game and did not generalize his approach to other games [18].

[2] The Nucleus is a solution concept in which a division is preferred where the maximum dissatisfaction is minimized

Early in the 20th century, important work has been done by Zermelo [42] who introduced an algorithm to find optimal strategies in fully observable games now known as Zermelos algorithm.

The modern field of game theory is considered to be founded by von Neumann and Morgenstein [38]. In their 1944 book they bundle the work published in the past two decades and provide a reference work for decades to come. Short after this Nash introduced the Nash equilibrium, the most used solution concept[3] in the analysis of games[24]. Although Nash has proven that every game has at least one Nash equilibrium, it remains a question how to find a Nash equilibrium.

With the increasing power of computers, scientist have been able to build computer programs which could beat the best human players in larger and more difficult games. An example is the Deep Blue computer which beat the world champion chess in 1997. Challenges currently worked on by scientist are building computer programs which can compete with top human players in two player Texas Hold'em poker [3] and Go [23]. Go and Poker are considered to be more difficult then chess because resp. the number of actions is bigger and the game is partial observable.

1.5 Thesis overview

In this thesis an algorithm will be introduced which learns to select the right actions from playing the same game multiple times. It can handle all games which can be represented in the normal form. The normal form can in principle represent any game with a discrete action space, but in practise is only used for complete observable games in which all agents make only one decision.

In the next chapter a more precise introduction to game theory is given, including an introduction to the most important algorithms for finding Nash equilibria. The third chapter introduces the concept of multiagent learning in games and defines the precise setting for the algorithm introduced in this thesis. Chapter four will introduce the proposed multiagent learning algorithm and chapter five discusses the experimental results. Chapter six discusses the relevance of this work and gives an outlook to further research which could build on this algorithm.

[3] A more thorough introduction of the Nash equilibrium will be given later, for now it suffices to think of a Nash equilibrium as optimal play

2. GAME THEORY

Game theory studies how rational agents should behave when playing a game.

Definition 1. A rational agent, is an agent which:

1. is only interested in maximizing its expected reward.
2. is flawless in its execution.
3. has enough computational power to complete the necessary computations.

To maximize its expected reward an agent has to select the right action, it is however depending on the actions selected by the other agents which action that is. Whether an agent expects his opponent to be stupid or extremely smart makes a big difference in the reward it expects after selecting an action. An action might give a high reward if the opponent reacts in a stupid way, but a very low reward if the opponent reacts appropriate. Making assumptions on the behaviour of an opponent is not without risk, an opponent could be deceiving the agent making him believe he will behave in a stupid fashion and then change to taking smart decisions to increase its own expected reward. To eliminate the possibility of agents which act non-optimal it is assumed that:

Assumption 1. *All agents are rational and it is common knowledge that they are.*

Where common knowledge is defined as something that everybody knows, everybody knows that everybody knows that everybody knows and so on ad infinitum. The common knowledge assumptions is critical even if all agents are acting rational: if one of the agents thinks that one of the other agents will play irrational (even if in fact he will not), this agent will adopt his actions to exploit the expected irrational behavior of this opponent and therefore behave differently than if the common knowledge assumption was not violated.

In the rest of this chapter is assumed that all players are rational and that this is common knowledge. The next section introduces the notation needed to discuss games in normal form. Then the concept of a Nash equilibrium is introduced, it is discussed how to find this equilibrium and a summary is given of the discussion on how valuable Nash equilibria is as a solution concept.

2.1 Normal form

The work in this thesis focusses on games represented in the normal form. Informally one could think of the following scenario: A group of agents meet that have never seen each other before nor will meet each other again, all agents write the action they select on a piece of paper and without showing it to the other agents hand it over to an oracle. After the oracle has received all papers, it distributes a certain number of coins among the agents depending on the actions selected by the agents, it is common knowledge among the agents how many coins each agent receives for every possible combination of actions the agents could hand over.[1]

A famous example game is the prisoners dilemma: two criminals are caught in a crime and brought to separate detention cells, both are offered two options: talk or keep silent. It is common knowledge between the agents that talking will lead to a higher penalty for the other agent and a reduction for the agent who talks, the reduction is smaller then the increase. Figure 2.1 summarizes the rewards received by each agent for each combination of actions, this table is called the reward matrix in two player games, and the reward tensor in games with more than two players.

	Silent	talk
Silent	$(-1, -1)$	$(-20, 0)$
Talk	$(0, -20)$	$(-10, -10)$

Fig. 2.1: The reward tensors (matrices) for the prisoners dilemma, agent 1 choses the row index and agent 2 the column index, the two actions together specify the reward received by the two agents (in years spend in prison). The first reward given is that of agent 1 and the second of agent 2

The intuition behind the model should now be clear, next a formal description of the model will be given and this will be illustrated with the example of the prisoners dilemma.

Definition 2. A normal form game is defined by:

- The number of agents N.
- Each agent i has a set of actions $A_i = \{a|0 < a \leq M_i\}$ to chose from, all agents pick their action at the same time without communication.
- The combination of all the individual actions is called the joint action $\bar{a} = [\bar{a}_1 \dots \bar{a}_N]'$ where $\bar{a}_i \in A_i$.
- The vector of all actions except those from agent i is denoted by $\bar{a}_{\neq i} = [a_1 \dots \bar{a}_{i-1}, \bar{a}_{i+1} \dots \bar{a}_N]$ and $(a, \bar{a}_{\neq i}) = [a_1 \dots \bar{a}_{i-1}, a, \bar{a}_{i+1} \dots \bar{a}_N] = \bar{a}$

[1] This model might seem somewhat limited, but is actually powerful enough to capture all games described in the introduction. To include the possibility of limited information and multiple decisions, every action an agent can pick has to correspond to a vector which specifies for every possible information set which action the agent should take. If all possible decision vectors are covered, the whole game is represented. A more in depth coverage of this conversion can be found in [28] or any other introduction text on game theory

- The set of all possible joint actions is denoted by A and the set of all possible joint actions excluding that of agent i is $A_{\neq i}$
- The reward tensor r_i for each player that specifies for every joint action, $\bar{a} \in A$, the reward, $r_i(\bar{a})$ for player i.

Note that the amount of numbers needed to represent a game grows exponential in the number of agents, if all agents have to make a binary choice there are $n2^n$ numbers needed to describe one game, this exponential relation limits all research in normal form games to a small number of agents.

The prisoners dilemma can be modelled as a normal form game: there are two agents, N =2, both agents can choose from two actions, $\forall i : M_i = 2$, the set of actions the agents can choose from is therefore: $\forall i : A_i = \{1, 2\}$, where 1 responds to deny everything and 2 with accepting the deal. The only thing left to define are the reward tensors for both players, since there are only two agents the tensors reduces to matrices: $\begin{pmatrix} -1 & -20 \\ 0 & -10 \end{pmatrix}$ and $\begin{pmatrix} -1 & 0 \\ -20 & -10 \end{pmatrix}$.

2.2 Selecting the right action

This section discusses which action a rational agent should select. To answer this question it will first be shown that it is relativly easy to find actions which are never optimal to select, this is discussed in the subsection on dominated strategies. To refine this technique even more it is demonstrated in the second subsection that agents sometimes need to randomize their actions. In the last section the most accepted solution concept is introduced which is called the Nash equilibrium, this concept is not without discussion and a summary of the discussion is provided.

2.2.1 Dominated strategies

The concept of dominated strategies is best explained by taking a closer look at the prisoners dilemma, the agents each could have the following thought experiment: suppose that the other agent does not talk with the police, what should I do? If the other agent selects silent, it is better to accept the deal and talk, this will save one year in prison. What if the other agent talks? I will go to jail anyway and can better talk as well to at least get a reduction in sentence. From this thought experiment both agents can conclude that no matter what the other agent does, it is always better to talk, both agents will talk and therefore go for 10 years in jail, a rather negative result (or positive from the police point of view).

The reasoning in the previous paragraph is called deletion of dominated actions: if for an action there is another action that for all possible combination of actions selected by the other agents performs better, this action should be removed from the analysis. Formally it is said that action a by agent i is dominated by action a' when:

$$\forall \bar{a}_{\neq i} : r_i(a', \bar{a}_{\neq i}) > r_i(a, \bar{a}_{\neq i}) \tag{2.1}$$

The deletion of dominated actions is an iterative process, when it is clear that one of the agents will never select an action because it is dominated, an action of another agent might become dominated, which was not dominated if the other player would select its dominated strategy, this is demonstrated in figure 2.2. Note that it is essential that the common knowledge assumption hold. If an agent can not be certain that the other agents will not select a dominated action it can not remove a conditional dominated action. It is not always possible to

	Left	Middle	Right
Up	$(1,0)$	$(1,2)$	$(0,1)$
Down	$(0,3)$	$(0,1)$	$(2,0)$

	Left	Middle
Up	$(1,0)$	$(1,2)$
Down	$(0,3)$	$(0,1)$

	Left	Middle
Up	$(1,0)$	$(1,2)$

	Middle
Up	$(1,2)$

Fig. 2.2: In the original payoff matrices ‘up‘ nor ‘down‘ can dominate the other, middle however can dominate right. If agent one assumes that player two will never play right, ’down’ becomes dominated after which player two can delete left, both players are now left with only one action

solve a game by iterative deletion of dominated strategies see figure 2.3. in this game it is not possible to delete any action as being dominated, if however player two introduces a new action, which when it is selected with 50% probability selects either action left or middle, this action will strictly dominate action right. The addition of this action does not change the game in any way, the new action is only present in the mind of the players and can be implemented by an agent which flips a coin and depending on the outcome writes down a different action on the paper passed to the oracle. Selecting an action based on a chance process is called a *mixed strategy*, opposed to a *pure strategy* in which one action is selected.

	Left	Middle	Right
Up	$(1,-1)$	$(-1,1)$	$(0,-0.1)$
Down	$(-1,1)$	$(1,-1)$	$(0,-0.1)$

	Left	Middle	Right	*Combination*
Up	$(1,-1)$	$(-1,1)$	$(0,-0.1)$	$(0,0)$
Down	$(-1,1)$	$(1,-1)$	$(0,-0.1)$	$(0,0)$

Fig. 2.3: None of the actions of player two is dominated by any of the other actions. However a new action which would select action left $\frac{1}{2}$ and action middle $\frac{1}{2}$ of the time would strictly dominate action right.

2.3 Mixed strategies

In the previous section it was shown by example that agent sometimes have to use mixed strategies, since the pure strategies used up till now can be represented as a mixed strategy from now on only mixed strategies will be used. In this section some extra notation needed for mixed strategies is introduced. Mixed strategies are not limited to the combination of only two actions or to an uniform distribution over the actions, but can be any probability distribution over the actions available to the agent. The mixed strategy of agent i is represented as a vector π_i of length M_i. Every element of the vector denotes the probability that the corresponding action is selected, π_{ia} is therefore the probability that agent i selects action a, $\pi_{\neq i}$ the joint strategies of all agents except agent i and $\pi = (\pi_i, \pi_{\neq i})$ denotes the joint strategies of all the players. Not all vectors are valid mixed strategies, it has to be a proper probability distribution, therefore all elements have to be between 0 and 1 and the vector has to sum up to one:

$$\begin{array}{l} \sum_{a=1}^{M_i} \pi_{ia} = 1 \\ \forall a : 0 \leq \pi_{ia} \leq 1 \end{array}$$

The set of actions for which hold that they have a none zero probability of being selected when playing a certain strategy π_i is called the *support* of i:

$$S(\pi_i) = \{a | a \in A_i, \pi_{ia} > 0\}$$

Given a certain joint strategy π the expected reward agent i receives when it would select action a is denoted by $u_{ia}(\pi)$,[2]:

$$u_{ia}(\pi) = \sum_{\bar{a}_{\neq i} \in A_{\neq i}} r_{i\bar{a}} \prod_{j=1, j \neq i}^{N} \pi_{j\bar{a}_j} \textbf{ s.t. } \bar{a} = (a, \bar{a}_{\neq i}) \tag{2.2}$$

Based on the expected reward received for each action the expected reward for agent i in the current joint strategy is:

$$U_i(\pi) = \sum_{a=1}^{M_i} \pi_{ia} u_{ia}(\pi) \tag{2.3}$$

The goal for each rational agent i is that the joint strategy selected by all agents maximize U_i.

2.4 Nash equilibrium

The deletion of dominated strategies can reduce the size of the payoff matrices and thereby simplify the game, it can however not always solve the game. Take the game called rock paper scissor in figure 2.4. Here it is not possible to remove any action by strict dominance by any strategy, additional tools are therefore needed to find the optimal strategy for the agents.

[2] This quantity resembles the Q-value in Reinforcement learning

	Paper	Rock	Scissor
Paper	$(0,0)$	$(1,-1)$	$(-1,1)$
Rock	$(-1,1)$	$(0,0)$	$(1,-1)$
Scissor	$(1,-1)$	$(-1,1)$	$(0,0)$

Fig. 2.4: The reward tensors (matrices) for the game of paper, rock, scissor. Two players make a sign with their hands denoting either paper, rock or scissor, if both players show the same sign it is a draw, otherwise the following ordering holds: paper beats rock, rock beats scissor and scissor beats paper.

An important concept in determining the right strategy for an agent is the best response:

Definition 3. Best response, a strategy that given the strategies of the other players gives the highest expected reward.

An action a is a best response when:

$$\underset{a \in A_i}{\operatorname{argmax}} : u_{ia}(\pi) \tag{2.4}$$

A strategy π_i is a best response to the joint strategy $\pi_{\neq i}$ when all actions in the support of π_i are a best response:

$$\forall a \in S(\pi_i) : u_{ia}(\pi) = \max_{a' \in A_i} u_{ia'}(\pi) \textbf{ s.t. } \pi = (\pi_i, \pi_{\neq i}) \tag{2.5}$$

From this and equation 2.3 it follows that agent i is playing a best response when the following equation holds:

$$U_i(\pi) \geq u_{ia}(\pi) \tag{2.6}$$

In his 1950 paper [24] Nash introduced the Nash equilibrium:

Definition 4. Nash equilibrium, a joint strategy in which all agents play a best response to the strategies of the other agents.

Which is equal to stating that any joint strategy π is a Nash equilibrium when holds that:

$$\forall i \forall a : U_i(\pi) \geq u_{ia}(\pi) \tag{2.7}$$

In paper-rock-scissor there is only one Nash equilibrium: both players select every action with chance $\frac{1}{3}$. To check that this is a Nash equilibrium the values of $U_i(\pi)$ and $U_{ia}(\pi)$ need to be compared, in the proposed Nash equilibrium they are 0 for all i and a proving that this is indeed a Nash equilibrium.

The Nash equilibrium is often presented as a solution concept in the sense that every agent should play a strategy which is part of a Nash equilibrium. It is however not without discussion since it does not give a complete solution in all games. In the article in which Nash introduces the Nash equilibrium he also

	Cinema	Ballet
Cinema	$(2, 1)$	$(-1, -1)$
Ballet	$(-1, -1)$	$(1, 2)$

Fig. 2.5: The battle of the sexes, a man and a woman want to go on a date together. Although both find it most important that they enjoy the company of the other, each prefers a different activity to undertake together.

proves that every game has at least one Nash equilibrium. If there is only one Nash equilibrium the game is solved and the agent know which strategy to pick. It could however be the case that multiple Nash equilibria exist in one game, it is then unclear which Nash equilibrium the agents should pick. This is called the Nash equilibrium selection problem.

A famous example is a game called battle of the sexes as shown in figure 2.5. This game has two Nash equilibria which are both in the pure strategies: (Cinema, Cinema) and (Ballet, Ballet). Since the agents can not communicate and have never met before nor will meet again, they have no idea which action the other will take and can not communicate with the other agent to reach an agreement. So despite the fact that they both want to play their part of a Nash equilibrium it is still unclear which action to pick.

Many attempts have been taken to refine the Nash equilibrium by introducing refined definitions of rationality, the general opinion is however that none of these give a satisfactionary solution[17].

In games where $N = 2$ and the rewards for any joint strategy sum up to a constant, the equilibrium selection problem is not relevant. In these games playing a Nash equilibrium guarantees a minimal payoff, independent of the Nash equilibrium selected by the other agent.

Proof. If the payoff of a player increases, that of the other agent decreases (constant sum). In a Nash equilibrium both players are playing a best response to the strategy of the other player neither agent can therefore increase its expected reward by picking another strategy. If the other player can not increase its expected payoff, the expected payoff of an agent playing a Nash equilibrium strategy can never be lower then the expected payoff he will get if both players played a Nash equilibrium. Q.E.D.

Therefore if multiple Nash equilibria exists the agents get the same payoff even if the other agent selects a different Nash equilibrium. In this setting the Nash equilibrium is also considered to be more valuable for another reason, if the other agents are not rational, an agent playing a Nash equilibrium is guaranteed a minimal payoff and is therefore playing a conservative strategy. This kind of guarantee is however not transferable to games with more then 2 agents.

2.5 Finding Nash equilibria

Despite the Nash equilibria selection problem in multiplayer games, a lot of effort has been put in algorithms capable of finding Nash equilibria in normal form games. In this section the algorithms used for solving the three major classes of games are discussed: solving a two player constant sum game as a Linear program, the Lemke-Howson algorithm for solving two player general sum games and the algorithms for solving N-player,M-action games.

2.5.1 Two player

Solving two player constant sum games is well understood, this problem can be cast into a linear program which can be solved with any standard linear program. The rest of this sub section explains how a linear program can be constructed that finds a Nash equilibrium for a normal form game, for a more complete introduction the reader is referred to [37] of which this section is a summary.

In the two player case the payoff tensors are actually two matrices, the expected reward for player i can therefore be calculated as:

$$u_i = \pi_2^T R_i \pi_1$$

Suppose that player one fixes its strategy then the goal of player two becomes:

$$\begin{array}{rl} \text{maximize} & \pi_2^T R_2 \pi_1 \\ \text{subject to:} & \sum_{a=1}^{M_2} \pi_{2a} = 1 \\ & \pi_{2a} \geq 0 \end{array}$$

Because the game is constant sum, $R_1 + R_2 = c$, the maximizing the reward of agent two is equal to minimizing the reward of agent one: $\pi_2^T R_1 \pi_1$. This is equal to selecting the minimum element from a vector and the solution is therefore a vector which is zero everywhere except for one position where it is one. It can thus be replaced by the unconstrained minimisation problem:

$$\min_a e_a^T R_1 \pi_1 \tag{2.8}$$

where e_a is the pure strategy selecting action a. Player 1 can thus predict the response by player 2 to any strategy he proposes, he will thus try to solve the following optimisation problem:

$$\begin{array}{rl} \text{maximize} & \min\limits_a e_a^T R_1 \pi_1 \\ \text{subject to:} & \sum_{a=1}^{M_1} \pi_{1a} = 1 \\ & \pi_{1j} \geq 0 \end{array}$$

This can be solved as a linear program which finds the maximum lowerbound, v, on the payoff of player 1:

$$\begin{array}{rl} \text{maximize} & v \\ \text{subject to:} & v < \min\limits_a e_a^T R_1 \pi_1 \\ & \sum_{a=1}^{M_1} \pi_{1a} = 1 \\ & \pi_{1a} \geq 0 \end{array}$$

The optimal strategy for player 2 can be found in a similar manner, it turns out that the linear program to find the strategy for player 2 is the dual of that of finding the optimal strategy for player 1. It should be clear to the reader that this efficient method can only be used in the two player constant sum case, because it needs the special relationship between the two payoff matrices and the fact that the calculation of the utilities from the payoff tensors and strategies can be written as a matrix multiplication.

The best known algorithm for finding Nash equilibria in two player general sum games is the Lemke Howson algorithm[19]. It has been published in 1964 and has since been the reference method with whom all work has been compared. In this algorithm the finding of Nash equilibria is considered as a special case of the Linear Complementarity Problem. The details of the algorithm are beyond the scope of this thesis, for a complete introduction the reader is referred to [34]. Only recently algorithms have been published which outperform the Lemke-Howson method [29, 32].

2.5.2 General case

Work on algorithms capable of finding Nash equilibria in games with more than two players has only appeared recently. The first method was published in 1987 by van der Laan et al. [36]. It builds a triangulated grid over the space of mixed strategy profiles and uses a path following algorithm to find a joint strategy for which holds that:

$$U_i(\pi) + \epsilon \geq u_{ia}(\pi) \tag{2.9}$$

It then refines the grid to find strategies with a smaller and smaller ϵ until it becomes zero.

The current state of the art is formed by the Govindan-Wilson algorithm [13] and the simple search method[29]. The simple search method exploits the observation that although it is hard to find a Nash equilibrium, it is relatively easy to check whether a Nash equilibrium exists with a certain support. The simple search method therefore starts searching all possible support sets, ordered from small to large, for a Nash equilibrium. This method turns out to work well especially for games often used in literature, in games generated using random numbers the advantages of the method declines.

The Govindan-Wilson method is based on a deep mathematical analysis of the problem structure and uses Newton method to find a gradient ascent to the Nash equilibria. In the original article the authors node that the method is hard to implement due the numerical stability issues that might arise, to deal with these accumulated errors the authors introduced a procedure they called wobbles, which might cause infinite loops.

Most algorithms discussed in this chapter, except for the simple search method (for which no reference implementation is available), are implemented in a software package called Gambit [21] . When the supplied algorithms where ran on the testset used in this thesis all algorithms failed on all but the smallest games. Our hypothesis is that in the case of the Govindan-Wilson algorithm this is caused due the wobble procedure. In the case of the Simplicial subdivision the cause could be that no real Nash equilibrium is near the early selected ϵ-Nash equilibrium. This is however a guess, the limited knowledge of the details of the algorithm by the author do not allow a more specific analysis.

2.6 Conclusion

The Nash equilibrium is a valuable concept. If it is common knowledge that all agents are rational all agents should play a strategy that is part of a Nash equilibrium. When looking at games with more then two player or games which are not constant sum this is not enough, there could be multiple Nash equilibria and the agents somehow have to coordinate their actions to select the same Nash equilibrium, it is unclear how this could be solved.

Even if the selection problem was solved a problem remains to compute Nash equilibria. Due to the exponential increase of the payoff matrices only small games, in the number of actions and agents, can be solved. Moreover the current methods for games with more than two players are inherently complex and hard to implement. The reference implementations suffer from numerical stability issues and are not able to solve large instances of the games in the test suit used in this thesis.

3. MULTIAGENT LEARNING IN GAMES

Multiagent learning (MAL) is almost as old as game theory and studies how agents can learn to play a game by repetitive play. The field is started by the publication of the fictitious play algorithm in the fifties[8] and recently two books have summarized the advances in the field [12, 40]. The research in MAL is diverse and strives many different goals. In a recent article Shoham et al.[33] suggested a list of the five most important goals in MAL:

Computational The computational agenda views multiagent learning algorithms as a computational means to find (Nash) equilibria in games. Compared to the Nash equilibrium finders discussed in the previous chapter these methods are usually not very efficient, but they can be easily understood and implemented.

Descriptive Finding formal models of learning that agree with behaviour observed in humans, animals, organisations and/or other agents. These works are on the boundary of Mathematics/Computer Science and other sciences like Economics, Biology, Psychology or Language.

Normative Determining which learning algorithms are in equilibrium. Two algorithms are in equilibrium when a rational agent will not switch to another learning algorithm if it knowns the learning algorithm of the other agents.

Prescriptive In prescriptive multiagent machine learning it is studied which learning algorithms maximizes the expected reward in an environment with other adapting agents. Agents might be using different learning algorithms and they might not be all equally effective.There is a big difference between the *cooperative* and *non-cooperative* case and these are considered to be different agendas. The cooperative case is generally considered more easy, since all agents have the same goal, there is no incentive to deceive each other.

In this thesis a new MAL algorithm will be introduced which focusses on the non-cooperative prescriptive goal, but is also a solution to the computational goal.

	Left	Right
Up	$(1,0)$	$(3,2)$
Down	$(2,1)$	$(4,0)$

Fig. 3.1: There is only one Nash equilibrium (Down,Left), however in a repetitive game rational learners should converge to a joint strategy where (Up,Right) has a big chance to occur.

3.1 Repetitive play

When an agent is learning it plays the same game multiple times against the same agents, this is called repetitive play. There are two different goals an agent could pursue in such a situation:

- Maximize the sum of the rewards received during all games played
- Maximize the expected reward in a single game

That these goals lead to different optimal strategies is illustrated in the game displayed in figure 3.1. In this game there is only one Nash equilibrium (Down,Left). Two agent which are interested in maximizing their reward in the current game will play this Nash equilibrium. Two agents interested in maximizing the sum of their rewards received during all the games played will adopt a joint strategy where (Right,Up) has a big chance to occur[1]. This gives both agents a higher expected payoff, Shoram et al. refer to this learning process as teaching, since agent one could teach agent two to play Right instead of Left by selecting Up most of the time.

3.2 On and offline learning

If the opponents are stable it depends on whether the learning takes place in an on or offline setting which of the two goals an agent pursues. In online learning the agent is put into the real world with an initial strategy and learns while playing the game. The agent is thus learning on the job and wants to maximize its total reward received. In offline learning the agent is put in a simulator with agents it is likely to have as an opponent when it is going to play the game for real. It can practice in this simulation how the other agents are adapting and search for the optimal strategy against these opponents. After the training the agent can either be placed in a setting in which it plays once against each opponent or in an online learning setting where the found strategy can be used as an initial strategy. In the first case the agent should optimize for a maximal expected reward in each game, in the latter it should maximize the total received reward. In a (semi) cooperative setting this could be used in the

[1] Such a mixed joint policy is the Nash equilibrium of the game which consists of an infinite number of repetitions of the single shot game

same way humans learn to cooperate as a team, they practise for example soccer by playing the game over and over again in a trainings mode and then play the game for real in a real match. In the non-cooperative setting a researcher could implement the learning algorithms it oughts likely to be used by the other agents and let its own algorithm learn against them.

In this thesis the goal is to build an offline learning algorithm which finds a optimal strategy for the one-shot game.

3.3 Evaluation of performance

How could the performance of two algorithms be compared? An important step in evaluating the performance is an exact goal definition, Shoram et al. [33] formulated the goal of a learning agent as:

> designing an optimal (or at least effective) agent for a given environment, where an environment consists of a game and the other agents (which may do some learning of their own).

The performance of an algorithm is therefore dependend of the games it has to play and the agents it will encounter as its opponents.

The games encountered in the literature are quite stable and it is therefore possible to have a reasonable prior. There are test suits available like the Gamut[26] set which contains a huge collection of games often studied in literature and form a good representation of the games such an agent will encounter. The algorithms of the other agents are however continuously changing and are an ongoing field of research. For general purpose algorithms one might be tempted to search for algorithms which perform well against all possible agents, this is however not possible due to the no-free-lunch theorem [39, 41]. If there is no good prior on the agents which can be encountered and it is not possible to design an algorithm which will perform better then any other algorithm averaged over all possible opponents, how could one evaluate two multiagent learners? This remains an open question, in the literature there is however general consensus that a good multiagent learning algorithm should at least satisfy the following minimal requirements [9]:

1. Against stationary[2]opponents the algorithm should converge to a best response strategy.

2. In self-play an algorithm should converge to a Nash equilibrium.

Requirement (1) is motivated by the fact that the single agent learners developed in the seventies, eg. q-learning or Sarsa-learning [35], are able to solve this case and any new suggested algorithm should therefore be able to achieve this.

Requirement (2) is motivated by the following observation: When finding an optimal strategy for an one-shot game an optimal learning algorithm learns to play the optimal response against the policies of the other agents. Therefore if all

[2] A stationary agent is an agent which does not change its strategy between two games.

agents are using an optimal learning algorithm, the strategies have to converge to a Nash equilibrium [15]. From this it follows that an optimal algorithm in self play has to converge to Nash.

These requirements are considered to be minimal in the sense that any algorithm that fails these requirements is considered to be a non satisfying learning algorithm. Satisfying these requirements is however not sufficient to be a good multiagent machine learning algorithm. This is often the case with the algorithms proposed in literature, although they generally satisfy the minimal requirements they have at least one of the following prerequisites:

1. The agents have to agree before the start of the algorithm to which Nash equilibrium they converge.
2. All agents can observe the strategies of all other agents.

Pre computing a Nash equilibrium requires the hard to implement methods of chapter 2 and if the computational power needed to find a Nash equilibrium is present, the agents could just as well play a Nash equilibrium strategy from the beginning. The second prerequisite is only a problem when using the learning algorithm online. In that setting the strategies of the other players are not visible, only their actions can be observed. In offline training however it is acceptable to observe the strategies of the other players, since everything is ran in a simulation and this information can be made available. As long as there are no algorithms that meet the minimum requirements and do not need a Nash equilibrium as input, the problem of evaluating the algorithms against different MAL algorithms can be postponed.

An interesting observation is that if an algorithm is found which meets minimal requirement (2) then this algorithm is also a solution for the computational goal, in this case it is off course essential that prerequisite (1) is not needed. It is therefore natural to suggest algorithms which are a solution to both research goals.

3.4 ϵ-Nash equilibrium

Due to the approximate nature of the multiagent learning algorithms they will most likely not converge to a true Nash equilibrium. It is therefore common to take an ϵ and then state that an joint strategy is an ϵ-Nash equilibrium iff:

$$\forall i \forall a : U_i(\pi) + \epsilon \geq u_{ia}(\pi)$$

Although there is no guarantee that there is an actual Nash equilibrium near an ϵ-Nash equilibrium in the strategy space, it is often used as an approximation where a smaller ϵ is considered a better approximation.

Nash equilibria are robust under affine transformations, $r'_{i\bar{a}} = \alpha_i(r_{i\bar{a}} + \beta_i)$ **s.t.** $\alpha_i \neq 0$, of the payoff matrices:

$$\begin{aligned} \max_a(u_{ia} - U_i) = 0 \quad &\Leftrightarrow \quad \max_a((u_{ia} + \beta_i) - (U_i + \beta_i)) = 0 \\ &\Leftrightarrow \quad \max_a(\alpha_i(u_{ia} + \beta_i) - \alpha_i(U_i + \beta_i)) = 0 \end{aligned}$$

However this does not hold for an ϵ-Nash equilibrium [4]:

$$\begin{aligned} \max_a(u_{ia} - U_i) < \epsilon \quad &\Leftrightarrow \quad \max_a((u_{ia} + \beta_i) - (U_i + \beta_i)) < \epsilon \\ &\not\Rightarrow \quad \max_a(\alpha_i(u_{ia} + \beta_i) - (\alpha_i U_i + \beta_i)) < \epsilon \end{aligned}$$

This is really awkward, its intuitive meaning would be that depending on how much money an agent possesses before the start of the game or whether the game is played in dollars or Euro, a certain set of strategies is an ϵ-Nash equilibrium or not. We therefore propose to normalize the payoff matrices of all agents before the analysis of the game:

$$\tilde{r}_{ia} = \frac{(r_{ia} - \min_a(r_{ia}))}{\max_a(r_{ia}) - \min_a(r_{ia})}$$

The normalized reward tensor, $\tilde{r}_i$, is not effected by the affine transformation applied in the original tensor and the affine transformation can therefore no longer affect whether a joint strategy is an ϵ-Nash equilibrium or not. Since the normalization is an affine transformation itself, it does not effect the real Nash equilibria.

3.5 Overview of MAL algorithms proposed in literature

Over the years many MAL algorithms have been proposed, many of them satisfy the minimal requirements in at least a well defined category of games, but none satisfies both requirements in all games and does not need a Nash equilibrium as input. In this section an overview of the most important algorithms will be given.

3.5.1 Awesome

Awesome is an acronym for Adapt When Everybody is Stationary, Otherwise Move to Equilibrium and is a direct implementation of the minimum requirements [9]. Every agent running this learning algorithm keeps a distribution of the actions selected by all the agents. Using this distribution it judges whether one of the agents is adopting its strategy, if all agent seem to be stationary the agent changes its strategy towards a best response against its current estimation of the other agents strategies. If however one of the agents appears to have changed it strategy, the agent moves its strategy towards a predefined Nash equilibrium and forgets all the data it has on the actions selected by the other agents. In order to keep synchronized with the other agents, every agents also checks if its own actions indicate a changing strategy, if so the agent also resets its learning progress (even if in fact its strategy is stationary, this is done to easy the proving of convergence).

The whole design of the algorithm is build around the two minimum requirements and the algorithm is therefore proven to meet them in all games, there are no experimental results included in the paper on how fast this convergence is met.

Although the algorithm meets the minimum requirements it is not going to perform very well against other agents which learn according to a different learning algorithm. It is hardwired in the algorithm to which Nash equilibrium it converges, suppose that all other agents converge to a strategy around a different Nash equilibrium, but never converge completely and therefore will never become stationary, this algorithm will then still converge to its predefined Nash equilibrium and not towards a best response. Furthermore it needs a Nash equilibrium as input in order to move towards it, which we identified as an undesirable property.

3.5.2 Regret Minimization

Regret quantifies how much better an agent would have done if it would have always selected action a instead of the strategies it used:

$$R_{ia}^T = \sum_{t=1}^{T} (\lambda(u_{ia}(\pi^t) - U_i(\pi^t)))$$

where $\lambda(x) = \left\{ \begin{array}{ll} 0 & x < 0 \\ x & x > 0 \end{array} \right.$, T is the amount of rounds played up till now and π^t is the strategy at time t. An action with a relative big regret should be chosen more often, the exact implementation of how many times a specific action should be selected is varied a lot between implementations. In this thesis one of the most simplest version is considered [3]. In this version the strategy at time T is chosen as follows:

$$\pi_{ia} = \frac{R_{ia}^T}{\sum_{a'=1}^{M_i} R_{ia'}^T}$$

Regret minimisation has been used successfully to build a competitive 2-player poker bot [43], but as will be shown in this thesis it does not converge to a Nash equilibrium in self-play very well when applied in N-player normal form games.

3.5.3 Wolf-PHC

Wolf-PHC, Win or Learn Fast - Policy Hill Climbing [5, 6], is a multiagent learning algorithm based on q-learning [35] which is proven to converge to a Nash equilibrium in the 2-player, 2-action case. Every agent keeps a vector which is an estimation of the expected reward received after selecting each action, this vector is called the Q-values and is updated by each agent after selecting action a:

$$\hat{Q}_i(a) \leftarrow (1-\alpha)\hat{Q}_i(a) + \alpha r_{i\bar{a}}$$

where $r_{i\bar{a}}$ is the observed received reward in this round, $\bar{a}$ the joint action selected and α the update speed which is typically in the order of 0.01. After the q-values are updated the policy is updated using:

$$\pi_{ia} \leftarrow \pi_{ia} + \left\{ \begin{array}{ll} \delta & \text{if } a = \operatorname{argmax}_{a'} \hat{Q}_i(a') \\ \frac{-\delta}{M_i - 1} & \end{array} \right.$$

The stepsize δ can have two predetermined values, if an agent is winning a bigger value is used then if an agent is losing. An agent is winning when, given the current estimations of the q-values, the average strategy up till this point:

$$\bar{\pi}_{ia} = \frac{\sum_{t=1}^{t=T} \pi_{ia}^t}{T}$$

is performing worse then the current strategy:

$$\sum_{a=1}^{a=M_i} \pi_{ia}\hat{Q}_i(a) > \sum_{a=1}^{a=M_i} \bar{\pi}_{ia}\hat{Q}_i(a)$$

The algorithm is proven to converge to a Nash equilibrium only in the 2-player, 2-action case, it is empirically shown that the algorithm converges to an ϵ-nash equilibrium in some simple example games. If the algorithm is however tested on a wider class of games it hardly converges, as is shown in the results section.

In the problem setting described in this thesis, it is acceptable to use the observed strategies of the other players, this should speed up the learning curve, in the algorithm implemented to compare performance we used the following update rule:

$$\hat{Q}_i(a) \leftarrow (1-\alpha)\hat{Q}_i(a) + \alpha u_{ia}(\pi)$$

Which is a direct implementation of equation 2.2.

3.5.4 RedValer

The RedValer, Replicator Dynamics with Variable Learning rate, algorithm is a based on the Wolf principle, but is proven to meet the minimum requirements [2]. The algorithm resembles the Wolf-PHC algorithm but uses a different update rule of its strategies, the update rule used is based on the replicator dynamics which updates the strategies according to the difference between the expected reward of each action and the current expected payoff:

$$\pi_{ia} \leftarrow \pi_{ia} l_{ia}(u_{ia}(\pi) - U_i(\pi))$$

where l_i is the learning rate. This learning rate is dependent on whether the other agents changed their strategy and if an action is selected more often then in a predefined Nash equilibrium π^*:

$$l_{ia} = \begin{cases} 1 & \text{if } \pi_{\neq i} \text{ is fixed} \\ \begin{cases} 1+\delta & \text{if } \pi_{ia} < \pi_{ia}^* \\ 1-\delta & \text{when } \pi_{ia} \geq \pi_{ia}^* \end{cases} & \text{otherwise} \end{cases}$$

after the update the strategy is normalized to form a proper probability distribution. The RedValer algorithm therefore needs both a precomputed Nash equilibrium and needs the strategies of all the players to be observable.

3.6 Summary

Multiagent learning is broad field in which many algorithms are proposed, each having a slightly different goal then another. The focus in this thesis is on applying multiagent learning in an offline fashion such that an agent learns the policy with the highest expected reward in each game given the learning algorithms of the other players. It is hard to evaluate this class of algorithms since it is not clear what a reasonable set of opponents would be. There is however general consensus in the literature that an important intermediate step would be a multiagent learning algorithm which converges to a best response when playing against stationary opponents and to a Nash equilibrium when playing against itself. There are several algorithms proposed in the literature, but none of them achieves these goal without needing a Nash equilibrium as input, this is however an undesirable property. If the game is simple enough to calculate the Nash equilibria, the question should be which Nash equilibria to play and there is no need for a multiagent learner. A MAL algorithm fulfilling the two minimal requirements and not needing a Nash equilibrium as input would therefore take science a step further to the goal of finding good MAL-algorithms.

A multiagent learning algorithm that is able to converge to a Nash equilibrium in self play and does not need a Nash equilibrium as input could also be used for an other goal: the computation of Nash equilibria. The current algorithms for finding Nash equilibria are very complex, while the multiagent learning algorithms are generally simple to understand. A multiagent approach to finding Nash equilibria would therefore work as a simple to implement, but slower alternative to the classic methods. If this is the goal then any learning algorithm which needs a Nash equilibrium as input is off course unsatisfactory.

4. WOLF-GRADIENT

In this chapter a new algorithm will be proposed for offline multiagent learning. The goal for such an algorithm is to maximize the expected reward in each game given the learning algorithms of the other agents. As discussed in the previous chapter the minimum requirements for such an algorithm are that it converges to a Nash equilibrium in selfplay and to a best response when playing against a stationary agent. Since there are no algorithms proposed yet that fulfill these requirements without needing a Nash equilibrium as input, this will be the focus of the proposed algorithm.

The algorithm is build around the Gain of each player (G_i). This quantifies how much the utility of an agent would improve if it would switch from its current strategy to a strategy which is a best response against the strategies of the other agents:

$$G_i(\pi) = \max_a u_{ia}(\pi) - U_i(\pi) \tag{4.1}$$

A related quantity is called maxGain, which is defined as the maximum Gain of all the agents:

$$G(\pi) = \max_i G_i(\pi) \tag{4.2}$$

The maxGain is closely related to the (ϵ)-Nash equilibrium. If the maxGain of a joint strategy is zero then it is a Nash equilibrium:

$$\forall i \forall a : U_i(\pi) \geq u_{ia}(\pi) \Leftrightarrow \max_i \max_a u_{ia}(\pi) - U_i(\pi) = 0 \tag{4.3}$$

and if it is smaller then ϵ then it is an ϵ-Nash equilibrium:

$$\forall i \forall a : U_i(\pi) + \epsilon \geq u_{ia}(\pi) \Leftrightarrow \max_i \max_a u_{ia}(\pi) - U_i(\pi) \leq \epsilon \tag{4.4}$$

Since it always holds that $G(\pi) \geq 0$, a Nash equilibrium, corresponds with a global minimum of the maxGain. A multiagent learning algorithm which in self play minimizes the gain and does not get stuck in a local optimum would therefore converge to a Nash equilibrium. This is exactly how the proposed algorithm is constructed.

The structure of the rest of this chapter is as follows: First it is demonstrated that if all agents use the gradient of their utility with respect to their strategy to update their strategy, the joint strategy does not converge to a Nash equilibrium. The method is then combined with the Wolf principle, the resulting method converges for some simple games, but in more complex games it could start to oscillate. In the last addition to the algorithm the step size of each player is limited, this results in a method that is converging to a Nash equilibrium in all games in our testset.

4.1 Pure gradient

Rational agents are only interested in maximizing their expected utility. This is convenient because an agent which is maximizing its utility is minimizing its gain under the assumption that the strategies of the other agents are stationary.

Proof. According to equation 2.2 $u_{ia}(\pi)$ does not change if the strategies of the other agents are stationary. If $u_{ia}(\pi)$ is constant it follows from equation 4.1 that maximizing $U_i(\pi)$ is equal to minimizing $G_i(\pi)$. Q.E.D.

It is therefore natural to let every agent maximize its utility after each game. A standard approach to maximizing a quantity is to update the variables, in this case the strategy of an individual agent, into the direction of the partial derivative, in this case: $\frac{\partial U_i}{\partial \pi_i} = \left[\frac{\partial U_i}{\partial \pi_{i1}} \dots \frac{\partial U_i}{\partial \pi_{iM_i}}\right] = [u_{i1} \dots u_{iM_i}]$. The update rule would then be:

$$\pi_i = \pi_i + \alpha \frac{\partial U_i}{\partial \pi_i} = \begin{bmatrix} \pi_{i1} + \alpha * u_{i1} \\ . \\ . \\ \pi_{iM_i} + \alpha * u_{iM_i} \end{bmatrix}$$

Where α is the stepsize which is typically $\ll 1$. After this update it is not guaranteed that the resulting strategy is a valid strategy. In most other algorithms this is solved by an operation which first guarantees that all elements are greater than zero and then normalizes the strategy vector. These procedures cause $\frac{\partial U_i}{\partial \pi_{ia}}$ to be discontinuous and can therefore not be used to update the strategies in all cases. In the proposed algorithm the strategy of agent i is therefore represented by an unconstrained vector σ_i, which is converted to a strategy by applying the softmax rule[7]:

$$\pi_{ia} = \frac{e^{\sigma_{ia}}}{\sum_{a'=1}^{M_i} e^{\sigma_{ia'}}} \qquad (4.5)$$

The advantage of representing the strategies in this space is that the derivative with respect to the expected utility, $\frac{\partial U_i}{\sigma \pi_i}$, still exists:

$$\begin{aligned}
\frac{\partial \pi_{ia}}{\partial \sigma_{ib}} &= \begin{cases} \frac{0\sum_{a'=1}^{M_i} e^{\sigma_{ia'}} - e^{\sigma_{ib}} e^{\sigma_{ia}}}{(\sum_{c=1}^{M_i} e^{\sigma_{ic}})^2} & b \neq a \\ \frac{e^{\sigma_{ia}} \sum_{a'=1}^{M_i} e^{\sigma_{ia'}} - e^{\sigma_{ib}} e^{\sigma_{ia}}}{(\sum_{c=1}^{M_i} e^{\sigma_{ic}})^2} & b = a \end{cases} \\
&= \begin{cases} \frac{-e^{\sigma_{ia}} e^{\sigma_{ib}}}{(\sum_{c=1}^{M_i} e^{\sigma_{ic}})^2} & b \neq a \\ \frac{e^{\sigma_{ia}} \sum_{a'=1}^{M_i} e^{\sigma_{ia'}}}{(\sum_{c=1}^{M_i} e^{\sigma_{ic}})^2} - \frac{e^{\sigma_{ib}} e^{\sigma_{ia}}}{(\sum_{c=1}^{M_i} e^{\sigma_{ic}})^2} & b = a \end{cases} \\
&= \begin{cases} \frac{-e^{\sigma_{ia}}}{(\sum_{c=1}^{M_i} e^{\sigma_{ic}})} \frac{e^{\sigma_{ib}}}{(\sum_{c=1}^{M_i} e^{\sigma_{ic}})} & b \neq a \\ \pi_{ia} - \pi_{ia}\pi_{ib} & b = a \end{cases} \\
&= \begin{cases} -\pi_{ia}\pi_{ib} & b \neq a \\ \pi_{ia}(1 - \pi_{ib}) & b = a \end{cases} \\
&= \delta_{ab}\pi_{ia} - \pi_{ia}\pi_{ib}
\end{aligned} \tag{4.6}$$

Where $\delta_{ab}$is one if $a = b$ and zero elsewhere, the complete derivative is:

$$\begin{aligned}
\frac{\partial U_i}{\partial \sigma_{ia}} &= \sum_{b=1}^{M_i} u_{ib} \frac{\partial \pi_{ib}}{\partial \sigma_{ia}} \\
&= \sum_{b=1}^{M_i} (u_{ib}\delta_{ab}\pi_{ib}) - \sum_{b=1}^{M_i} (\pi_{ia}\pi_{ib}u_{ib}) \\
&= \pi_{ia}u_{ia} - \pi_{ia} \sum_{b=1}^{M_i} (\pi_{ib}u_{ib}) \\
&= \pi_{ia}(u_{ia} - U_i)
\end{aligned} \tag{4.7}$$

Which gives us an intuitive appealing format: reduce the probability of actions for which the expected payoff is lower then that of the current strategy and increase the probability of actions for which it is higher.

If all agents implement this learning rule, the assumption that $\pi_{\neq i}$ is stationary is off course violated. Learning according to this algorithm is therefore not guaranteed to converge to a Nash equilibrium. If two agents use this update rule in self-play this leads to circular behaviour in for example the game matching pennies (see figure 4.1 for an explanation of the game). In figure 4.2 the joint strategy of two agents updating their strategy according to $\frac{\partial U_i}{\sigma_{ia}}$ is visualized by letting one axe denote the probability that agent one selects head and the other axe that agent two selects head. In this two dimensional plain the direction of the update of the joint strategy is always perpendicular to the Nash equilibrium which is in the center. The joint strategy should therefore make a perfect circle, the update steps are however not infinite small and the joint strategy therefore spirals outwards, away from the Nash equilibrium.

	Head	Tails
Paper	$(1,-1)$	$(-1,1)$
Rock	$(-1,1)$	$(1,-1)$

Fig. 4.1: The reward tensors (matrices) for the game of matching pennies. Two players show a side of a coin, if both players show the same sign they player one wins, otherwise player two wins.

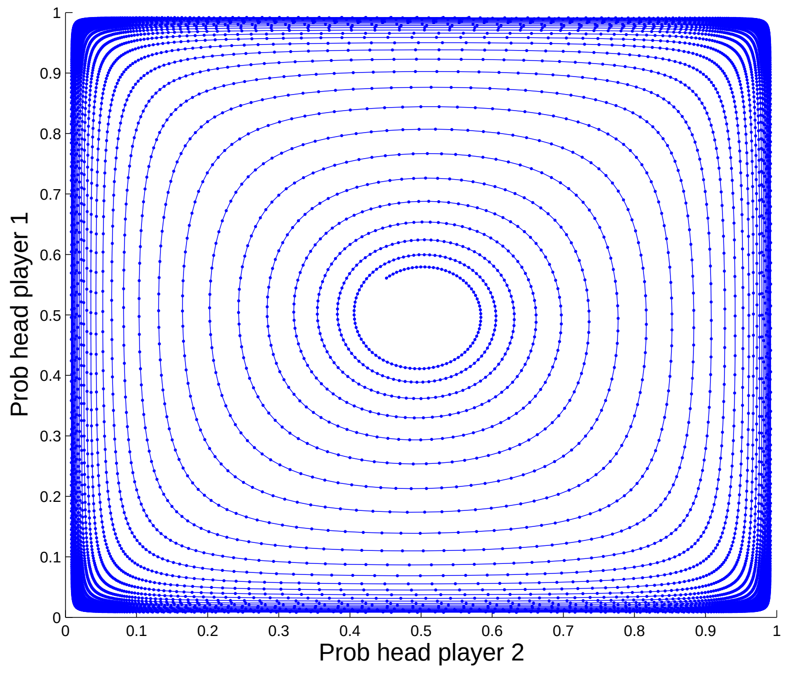

Fig. 4.2: The horizontal/vertical axe gives the probability that resp. player two/one select Heads. The players all update their strategy according to the pure gradient. The joint gradient is always perpendicular to the Nash equilibrium, but since the updates are performed in finite steps the strategies circle outwards instead of making a perfect circle around the Nash equilibrium

4.2 *Wolf principle*

If the agents all update their strategies according to the derivative, the policies do not converge in even the simplest game. The update rule is therefore adjusted such that only the agent with the biggest gain updates its policy, the policies of the others are unchanged. This is motivated by two observations:

- The goal is to minimize the maxGain, the increase of the utility of the player with the highest gain will reduce this quantity (as long as the gain of the other players does not increase to a level which is higher then the current maximum gain).
- This update rule is an extension to the Wolf principle, a winning player is a player that is already playing the best response against its opponents, this is equal to having a low gain. The player with the highest gain is therefore the losing player and should make the biggest move.

The proposed update equation is therefore:

$$\sigma_{ia} \Leftarrow \begin{cases} \sigma_{ia} + \alpha\pi_{ia}(u_{ia} - U_i) & \text{if } i = \text{argmax}_i\, G_i \\ \sigma_{ia} & \text{otherwise} \end{cases}$$

For this algorithm to work the reward matrices have to be normalized before the algorithm is ran (just as with ϵ-nash equilibria), otherwise the gains of the different agents are incomparable. Suppose that one of the matching pennies players receives its reward in roubles, its reward matrix is effectively multiplied by 1000 and its gain will therefore always be bigger then that of an agent playing in dollars. If the two reward matrices are normalized before the algorithm is ran, this is no longer a problem.

A learning algorithm using this update rule converges to a Nash equilibrium in self-play in for example the two player matching pennies game. In games with more then two players the joint policy no longer converges. As an example the algorithm has been ran on a N-player extension of the matching pennies game. This game uses the following rule set is used:

- There is an uneven number of agents which are ordered in a circle.
- All agents show either head or tail at the same time.
- If an agent shows the same sign as the agent next to it, it gets a negative reward, otherwise it gets a positive reward.

There is only one Nash equilibrium which is when all agents select one the actions with chance $\frac{1}{2}$. In figure 4.3 it is shown that in this three player game the maxGain moves to an asymptote. This is caused by the phenomenon that the gain of another agent rises more then the gain of the previous maxGain agent drops.

4.3 Stepsize

Although an adoption of the strategy of the maxGain player will decrease the gain of this player, it might increase the gain of another agent causing the maxGain to rize instead of decrease. The step taken by the agent with the

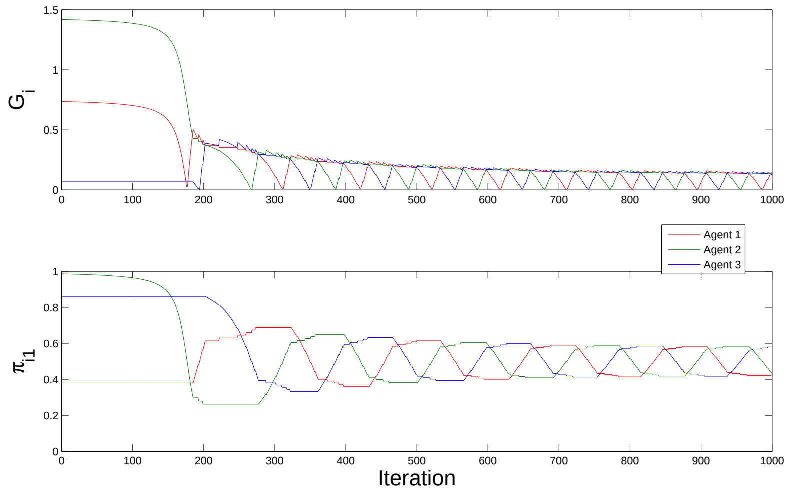

Fig. 4.3: Three agents are playing matching pennies, the horizontal axe gives the learning iteration, the vertical axe represents the Gain of each player and probability that each player selects head. Although the agent with the maximum gain reduces its gain, the gain of the agent which is previous in line causes the maximum gain to rise instead of drop.

biggest gain therefore has to be limited to reduce the maxGain. Instead of using a fixed α, the agent uses the line search depicted in algorithm [22] to find the biggest α that decreases the maxGain with at least $\delta_{G_{min}}$ percent. To guarantee that the algorithm finishes there is minimum value defined for alpha α_{min}, if no satisfactory *alpha* is found which is bigger then this value the algorithm is stuck in a local minima, in algorithm 1 the complete search algorithm is defined. It uses the update function which is defined in algorithm 2.

Algorithm 1 LineSearch(σ,i)

```
1: $\alpha = \alpha_{start}$
2: while $\alpha > \alpha_{min}$ do
3:     $\sigma' =$update$(\sigma, i, \alpha)$
4:     if $G(\sigma'_i) < (1.0 - \delta_{G_{min}})G(\sigma_i)$ then
5:         return $\sigma'$
6:     end if
7:     $\alpha = \alpha/2$
8: end while
9: return None
```

Algorithm 2 update(σ,i,α)

```
1: return $\forall j : \sigma'_j = \begin{cases} \sigma_j + \alpha\frac{\partial U_j}{\partial \sigma_j} & i = j \\ \sigma_j & \text{otherwise} \end{cases}$
```

The proposed line search algorithm can fail if it is impossible for the player with the maximum gain to move and decrease the maximum gain with a minimal amount. In this situation it could be possible that one of the other agents can make a move which would decrease the maximum gain. If none of the players can make a move the algorithm is stuck in a local optimum, to get out of this optimum the player with the biggest gain takes a considerable step to increase its expected utility. The pseudocode for this search algorithm is implemented in algorithm 3.

Algorithm 3 in not yet a learning algorithm, it describes how the joint policy should be updated, while a learning algorithm should be run by each individual agent updating only the policy of that agent. To turn this algorithm into a proper leaning algorithm the algorithm is ran in parallel in each agent, where each agent updates their strategy to that in the new joint policy.

To guarantee that the algorithm converges to a best response in case one of the agents is playing a stationary strategy, every agents checks whether all agents that should have changed their strategy did so. If an agent did not change its strategy when it should have, it is considered a stationary agent. A stationary is seen as a part of the environment, it is removed from the list of players and the agents update their rewards tensors as if this agent will always be playing this strategy.

Algorithm 3 FindPlayerToMove(σ)

```
1: sortedListOfPlayers = sortOnGain($\sigma$)
2: while not Empty(sortedListOfPlayers) do
3:     $i$ = pop(sortedListOfPlayers)
4:     updatedJointStrategy = LineSearch($\sigma$,$i$)
5:     if updatedJointStrategy $\neq$ None then
6:         return updatedJointStrategy
7:     end if
8: end while
9: $i = \text{argmax}_i\, G_i$
10: $\sigma' = \text{update}(\sigma, i, \alpha_{stuck})$
11: return $\sigma'$
```

These two steps are incorporated in algorithm 4 which is the final algorithm proposed in this thesis.

Algorithm 4 learnStep(σ,i)

```
Require: $\delta_{G_{min}}$, $\alpha_{start}$, $\alpha_{stuck}$, normalized($r_i$), noDominatedStrategies($r_i$)
1: Remove stationary players from the analysis
2: $\sigma'$ = FindPlayerToMove($\sigma$)
3: return $\sigma'_i$
```

4.4 Summary

In this chapter a new MAL-learning algorithm has been introduced. In main focus is to converge to a Nash equilibrium in self-play. The basis of the algorithm is formed by a quantity called gain, which quantifies how much an agent would profit if it would switch to the best response against the current strategies of the other agents.

The basic outline of the algorithms is as follows: an agent only adopts its strategy if it has the highest gain of all agents. If this is the case it updates its strategy in the direction of the gradient of its current strategy with the biggest stepsize such that the maxGain will decrease with a minimum percentage. If this is not possible one of the other agents will try to make such a move. If none of the agents can make such a move the agent with the highest gain will make a move in the direction of its gradient with a predefined stepsize. Before an agent starts to adjust its strategy it checks whether all agents have adopted their strategy like they should have in the previous round. If this is not the case the agent is considered to be stable and the agent is ignored, this is implemented by updating the reward tensors such that they reflect the rewards the agents will receive if this agent never changes its strategy and ignoring the strategy of that agent from now on.

The convergence properties of this algorithm are discussed in the next chapter.

5. EXPERIMENTAL RESULTS

In the previous chapter a multiagent learning algorithm called Wolf-Gradient is proposed. The goal of this algorithm is to learn an optimal strategy against other (learning) agents. The learning takes place in an offline simulation, the strategies are therefore visible and the simulation can be restarted. After several restarts the best joint strategy is selected to play in an online environment. There are two main demands for an algorithm with this goal:

- When playing against stationary agents it should converge to a best response
- In self play it should converge to a Nash equilibrium.

Since the first demand is always satisfied by the design of the algorithm, the main point of this chapter is to compare the convergence properties of the Wolf-Gradient algorithm empirically with that of algorithms previously proposed in the literature.

For every game tested the algorithms are ran 9 times after which the joint strategy with the smallest maxGain is selected, these random restarts help algorithms which get stuck in local optima. The games are generated using game generators. These generators are usually defined for a variable number of agents and actions per agent (the size of a game) and use some sort of random process to generate different instances of a game with the same size. To get a good impression of the performance of every algorithm for different sizes of games, a test set has been composed of several game generators and for each generator several sizes have been selected. Not all sizes are valid for all generators and the sizes used therefore differ per generator. For each combination of generator and game size ten game instantiations are generated. On each of these ten instances every algorithm is run 9 times for 25000 iterations. Each run of an algorithm gives a certain joint policy with a certain maxGain, after nine restarts the joint strategy is selected with smallest maxGain. This gives a maxGain for each algorithm for each instance of a game, to compare the performance of the algorithm for this particular combination of size and generator, the maximum of these maxGains is taken for each algorithm. This gives an indication of the worst case performance of the algorithm. If this maximum over the maxGains is zero, an algorithm always converges to a perfect Nash equilibrium for this type of game. In figure 5.1 there is a schematic representation of the construction of the testset.

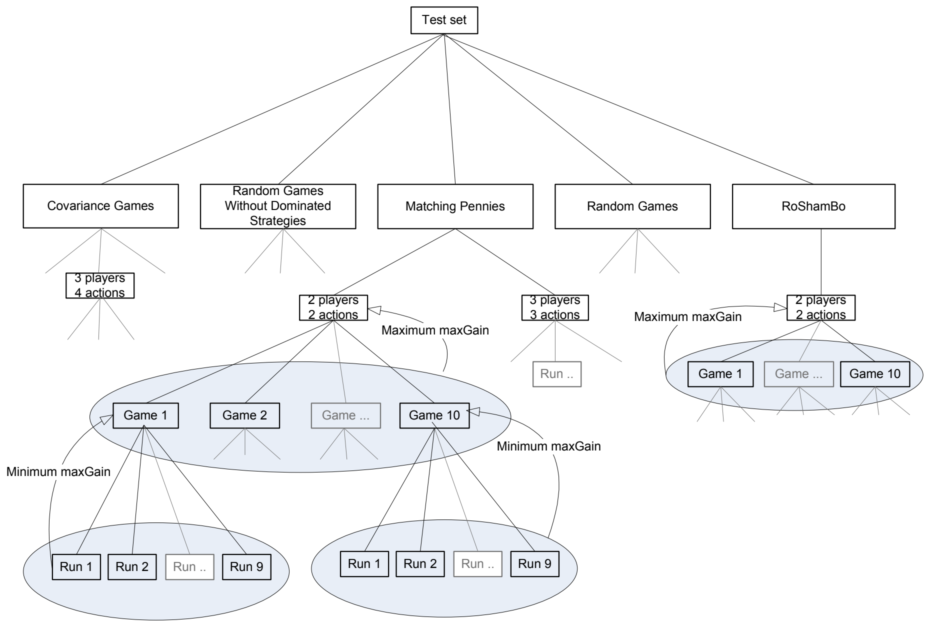

Fig. 5.1: Schematic overview of the testset. For each game generator there are 10 different examples generated, each algorithm is ran nine times on each example for 25000 runs. From the nine generated strategy the one with the smallest maxGain is selected. To quantify the performance of an algorithm the worst case is considered, from the 10 maxGains of these final strategies the maximum is taken. The algorithm which scores the lowest on this quantity is considered to have the best convergence properties of all algorithms for this class of games.

5.1 The algorithms and the parameters

The algorithms to which the Wolf-Gradient algorithm is compared should not need a Nash equilibrium as input, since this is the major achievement of this algorithm. For each of these algorithms there is a paragraph in which we discuss the used parameters. There are three algorithms which fulfill this requirement:

Wolf-PHC The Wolf-PHC algorithm needs two parameters, after some experimentation the following settings seem to give the best performance:

- $\delta_l = 0.001$
- $\delta_w = 0.0003$

Wolf-PHC-Gain this is identical to the Wolf-PHC algorithm except that the losing agent is picked by determining the agents with the largest gain. This algorithm is introduced to test the performance of the losing agent selection separately of the other ideas introduced in this thesis. The parameters used are the same as in the Wolf-PHC algorithm.

Regret-minimization which has no free parameters.

Wolf-Gradient The parameters for the Wolf-Gradient algorithm are:

- $\alpha_{start} = 30$
- $\alpha_{stuck} = 3$
- $\alpha_{min} = 0.3$
- $\delta G_{min} = 0.01$

5.2 Test games

In this section the choice of game generators is explained and the details of each generator discussed. For each generator plots are included which compare the performance of the algorithms.

The most common test set used in literature to empirically test algorithms for normal form games is the Gamut test set [26]. This test set contains a huge collection of games collected from the last fifty years of research. Most games in this test set contain pure Nash equilibria and can therefore be solved easily by enumerating all possible pure joint strategies. Once the Nash equilibria are known only the selection problem remains. These games are therefore not very interesting for our algorithms and are therefore not considered[1]. To compare the learning algorithms a selection has been made from the games available from

[1] The proposed algorithm has been tested on games with pure strategies informally and converged in all cases.

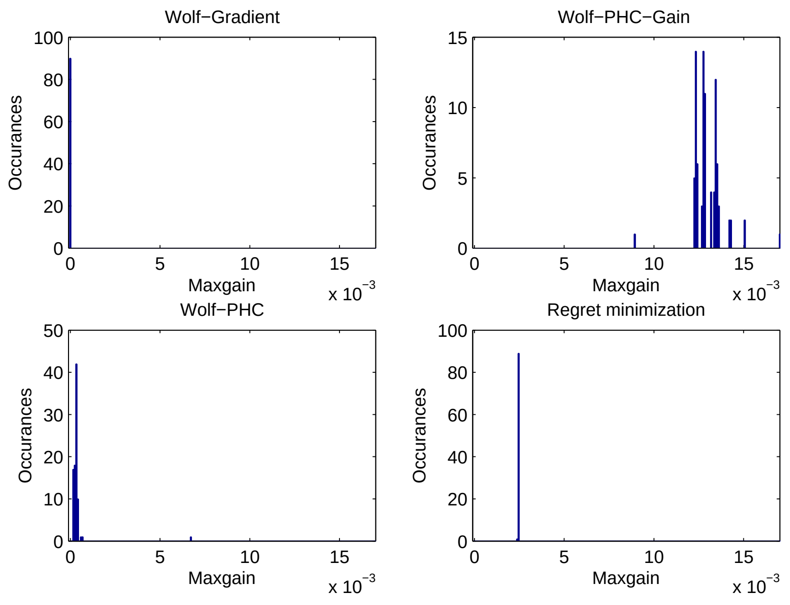

Fig. 5.2: Histogram of the maxGain after 90 simulations of 25000 rounds in two player matching pennies. Wolf-Gradient and Regret minimization always converge to a perfect Nash equilibrium. To give a better insight in the maxGains of the found joint strategies, the maxGains of all the strategies found after each restart are used for this histogram.

the Gamut set and several other games have been added to give a more balanced view. The games in the test set are discussed in the separate subsections together with the performance of each algorithm.

5.2.1 Matching pennies

The matching pennies game has been introduced in section 4.1. The two player version is part of the Gamut suite and the N-player version has been introduced by the author. The game has only one Nash equilibrium and that is when all agents select head with probability $\frac{1}{2}$. Since the game is very simple and only two versions are computational tractable this game can be analysed in more detail. For both the two and three player game a histogram of the normalized gains observed at the end of every simulation is displayed in resp. figure 5.2 and figure 5.3. Note that for these histograms the nine restarts performed on each game are not combined by taking the minimum, but are all included individually.

In the two player case the Wolf-Gradient method is the only algorithm which converges to a true Nash equilibrium, all other algorithms converge to ϵ-Nash

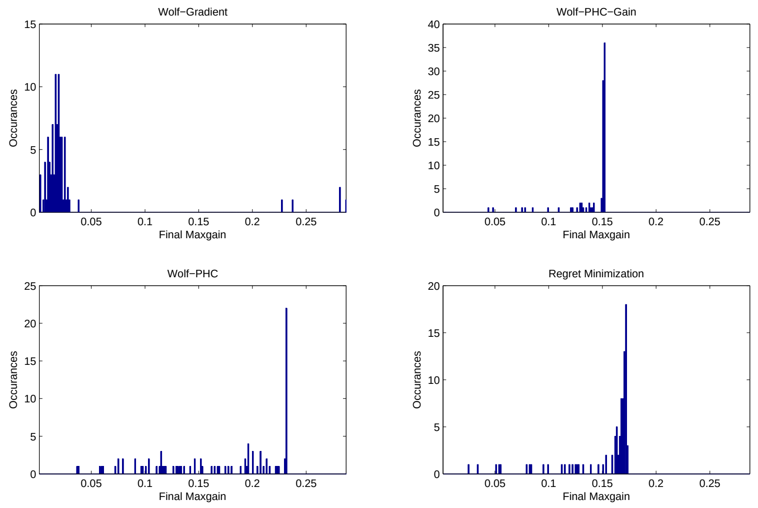

Fig. 5.3: Histogram of the found maxgains in three player matching pennies. Again the maxGains after each random restart are used. Except for some outliers the Wolf-Gradient algorithm has the best convergence properties of all algorithms discussed.

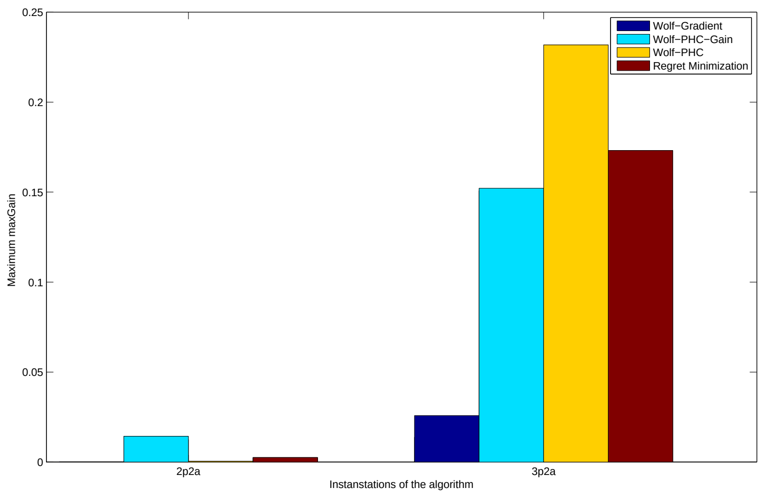

Fig. 5.4: This plot shows, for the matching pennies game, the maximum max-Gain of all joint strategies generated by letting every algorithm learn 25000 iteration after each of the nine random restarts and selecting the joint strategy with the smallest maxGain. The labels of the x-axe denote the size of the games, 3p2a denotes 2 players 2 actions.

equilibria. Despite the fact the Wolf-PHC algorithm is proven to converge in all 2-player, 2-actions games, it does not converge in this simulation. This anomaly is caused by the fact that the proof is based on infinitely small steps, while in practise the steps taken are finite.

In three player matching pennies all other algorithms again converge asymptotically to a Nash equilibrium, but never reach it. In this game the wolf-gradient method also does not converge to a perfect Nash equilibrium in all runs. The convergence is however much better then that of the other algorithms, except for some outliers. The outliers appear when one of the agents at some point in time selects an almost pure strategy, the derivative of the soft-Max then becomes infinitely small and the strategy of this agent therefore does not change for many iterations. While this agent is stuck in a pure strategy, the other agents adopt their strategy to a best response which is also a pure strategy. Once all agents are selecting pure strategies they start oscillating in turn changing from one pure strategy to another, this stops the algorithm from converging. This is fixed by restarting the algorithm.

If the best joint strategy is selected after the nine random restarts the max-Gain of the worst joint strategy is still not zero, but it is significant smaller then that of the other algorithms see figure 5.4. With such a small maxGain it is save to assume that the algorithm converged in all cases.

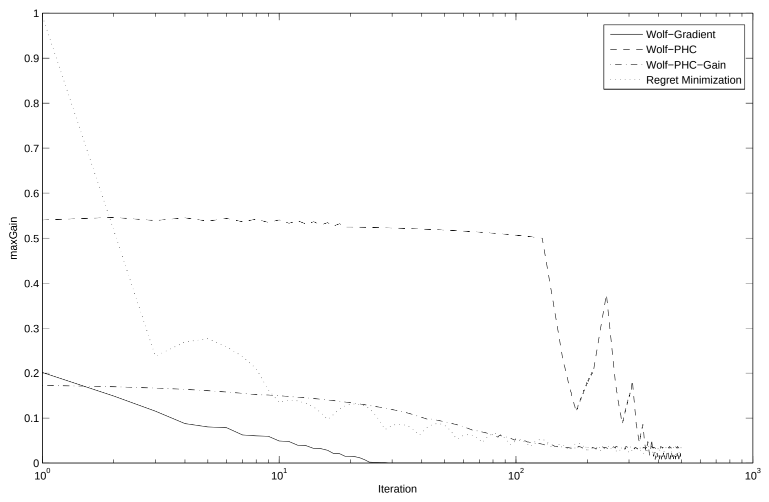

Fig. 5.5: Trace of the maxGains in a typical convergence of two player matching pennies. The speed of convergence of Wolf-Gradient is much higher then of the other algorithms. It can also be seen that the other algorithms do not truly converge to Nash but to some ϵ-Nash equilibrium

When the speed of convergence is compared, the Wolf-Gradient algorithm converges always much faster then the other algorithms. See figure 5.5 for a typical trace of the maxGains of all algorithms during the learning phase. In this plot it is also visualized that the other algorithms do not truly converge to Nash but get stuck at an ϵ-Nash equilibrium.

5.2.2 RoShamBo

RoShamBo, also known as Paper Rock scissor, is a well known game often played by humans. Although most people see the game as simple and the optimal strategy is known (play completely random), there exists official world championships for humans[2] and computers. In the human league the reward for the best player is \$10.000 and the game is therefore taken more serious by some people then one might expect.

The rules of the game are simple, both agents select, at the same time, one of three symbols: Paper, rock or scissor. If both select the same it is a draw, otherwise: rock beat scissor, paper beats rock and scissor beats paper. The Nash equilibrium is for both players to play each of the three actions with probability $\frac{1}{3}$.

[2] http://www.worldrps.com/

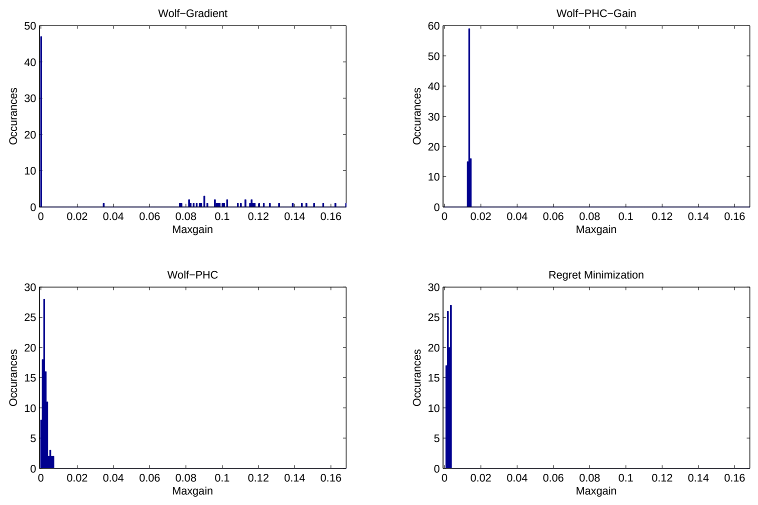

Fig. 5.6: Histogram of the found maxgains in RoShamBo after each random restart. The Wolf-Gradient algorithm converges perfect in half the cases and to some quite big value in the other cases. If a results from the random restarts are however combined, it converges always perfectly.

How can the game be interesting to play if the optimal solution is known? Suppose that a tournament is played between three agents, every agent plays a number of head to head games against all other agents, the agent which wins the most games wins. Suppose one agent is playing the Nash equilibrium, and the other two are trying to exploit the strategy of their opponent and therefore play a non-Nash strategy. In this competition the Nash equilibrium player will win exactly half of the games played. The other agent will also win half the games played against the Nash equilibrium player, but in the games played against each other one will win more games and the other will lose more then half the games. The Nash equilibrium player will therefore always end in the middle of the ranking and the other two will end first or last. It is therefore never interesting to play Nash and such a competition will therefore be an interesting setting for MAL-algorithms [3]. In figure 5.6 the results are shown in the same histogram as for the matching pennies game. Again it is shown that the Wolf-Gradient method converges to a perfect Nash equilibrium and the other algorithms do not. In the cases where the Wolf-Gradient does not converge to a Nash-equilibrium it oscillates between pure strategies. If the algorithms are however restarted several times the Wolf-Gradient method is again the only algorithm which converges to perfect Nash equilibrium see figure 5.7.

[3] The computer programs which are currently employed at these kind of competitions are handcrafted for this specific problem and not based on general multi-agent learning algorithms

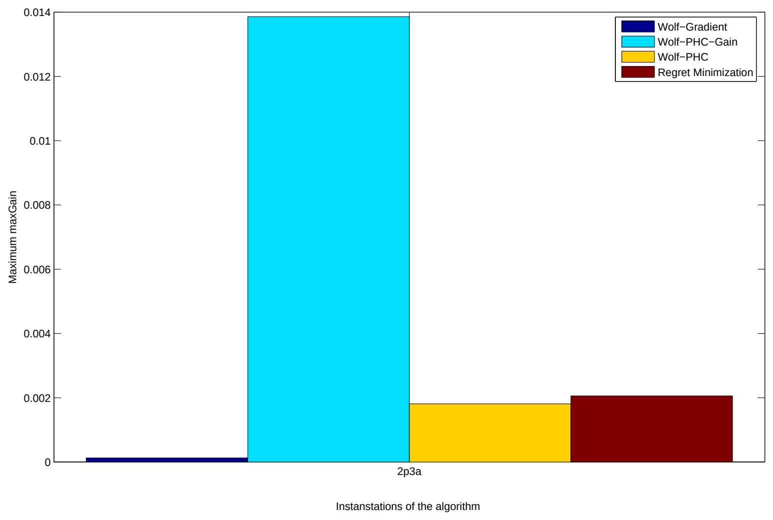

Fig. 5.7: Overview of the maximum maxgains found in RoShamBo, when all algorithms are restarted nine times. In this case the Wolf-Gradient algorithm always converges almost perfectly, $\epsilon < 10^-4$, while none of the other algorithms does.

5.2.3 Random games

Random games are created by generating reward tensors for each agent using an uniform random number generator, this type of games is a standard part of the Gamut testsuit and it is one of the most popular games to test algorithms on. Random games are not without controversy, at first look they might look like the most fair games to test algorithms on, but a common argument against this is that interesting games encountered in real life form a very specific distribution and performance on uniform distributed games is therefore not representative for the performance in reallife games. Despite this controversy random games are still often used to test algorithms on.

We have selected several computational tractable size of random games to form our testset. The size of the games might look small, but one has to remember that a three player four action game is represented by $3 \times 4^3 = 192$ numbers for which in each round the utilities have to be calculated which denotes $3 \times 192 = 576$ operations, to perform the linesearch this has to be repeated several times per round. Repeating these calculations for 25000 rounds gives an operation which takes about a hour per game. In figure 5.8 an overview is given of the maximum maxGain of the found joint strategy for every game size. Some interesting observations can be made, first of all the Wolf-Gradient algorithm finds a Nash equilibrium in all cases except the two player four action case, while the other algorithms only converge in the two player three action games.

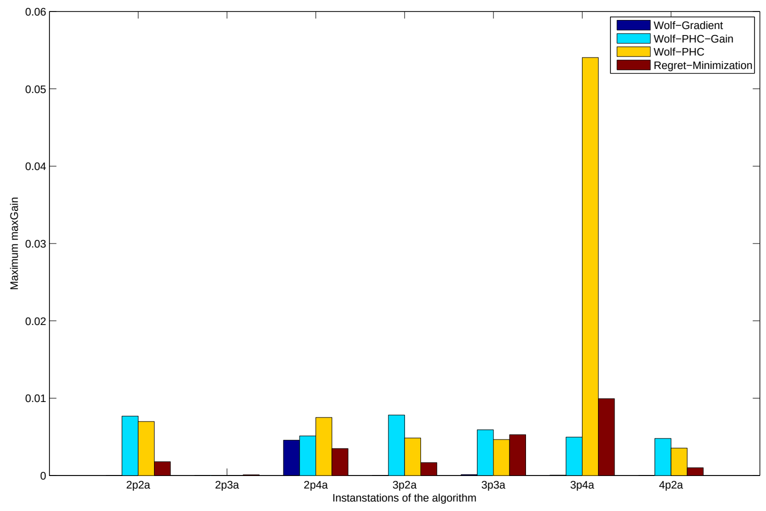

Fig. 5.8: An overview of the maximum maxGain of the joint strategy each algorithm came up with for different sizes of random games. After nine random restarts and 25000 iterations after each restart.

That all algorithms converge in the two player four action case is caused by coincidence, all games generated contain only one pure Nash equilibrium which can be found by deletion of dominated strategies. Since all dominated strategies are removed in the normalization process, these games contain only one action for each agent. In the two player four action case the Wolf-Gradient does not converge in the worst case because in two games out of the 10 generated, it gets stuck in a pure policy after each restart.

5.2.4 Random games without dominated strategies

Many games generated using an uniform distribution contain dominated strategies and often even contain a pure Nash equilibrium, properties we wanted to avoid in our test set. In the this section a new type of game is introduced which we think is a better alternative to the completely random games and should be used instead. This generator generates a game using the same uniform distribution as the uniform random generator, but rejects any tensor that contains a dominated strategy. This process is continued until a reward tensor is generated for each agent which contains no dominated strategies. The algorithm is tested using the same method as the Random games, the results are displayed in figure 5.9. Again the Wolf-Gradient algorithm converges in almost all games, the big exception is the two player four action game. Again a more detailed look at

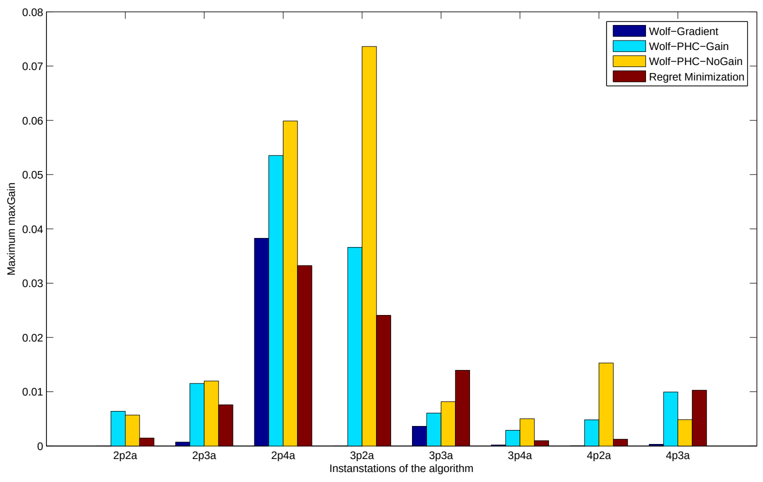

Fig. 5.9: An overview of the worst joint strategy each algorithm came up with for different sizes of random games without dominated strategies.

the two player four action case shows that the algorithm converges in almost all games generated except one, where it gets stuck in a pure strategy after each restart.

5.2.5 Covariance games

In a covariance game the payoff tensors of each player are generated using a normal distribution with the restriction that the covariance between the rewards of two players given the joint action has a fixed value. A positive covariance indicates that if one agent gets a relatively big reward for a certain joint action, the chance that the other agents get a relatively big reward for the same joint action increases. A negative covariance indicates that a bigger reward for one agent indicates a smaller reward for the other agent. This gives a sliding scale between a fully cooperative game, covariance = 1, and a game in which the rewards are maximal opposite, covariance = $\frac{-1}{N-1}$. In this test set the covariance is set to $\frac{-1}{N-1}$ since the goal is to find a learning algorithm for the non-cooperative case. Again the Wolf-Gradient algorithm converges in all games except for the two player four action case and again the Wolf-Gradient algorithm converges in all games generated excepts one. In the game instance in which it does not converge at least one agent gets stuck in a pure strategy after each initialization.

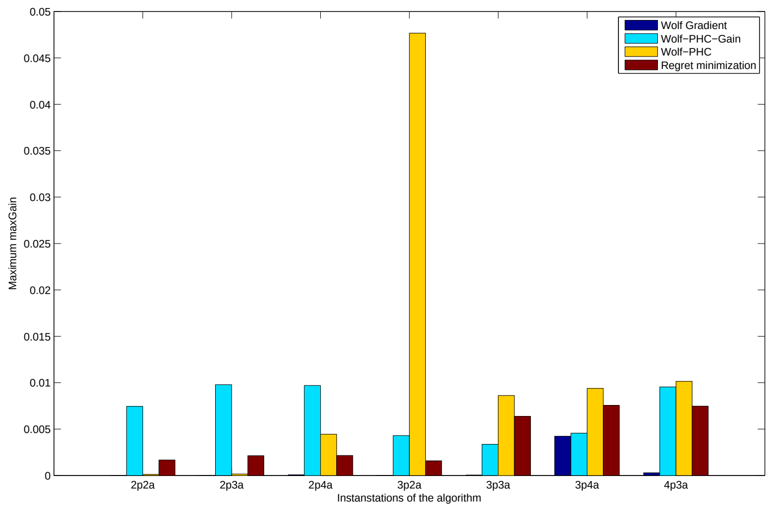

Fig. 5.10: An overview of the maxGain of the worst joint strategy each algorithm came up with for the ten games covariance games of different sizes, after nine restarts and 25000 rounds after each restart.

5.3 Summary

The Wolf-Gradient method converges to a true Nash-equilibrium in a wide variety of games, this is not achieved by any of the other algorithms proposed in literature so far. The gradient of the softmax transformation becomes zero, when one of the agents selects a pure strategy, this causes the algorithm to take an infinite amount of time to move to move to a different strategy and the algorithm therefore gets stuck in practise. This is the only cause of the algorithm to fail.

6. CONCLUSION

In this thesis a new multiagent learning (MAL) algorithm is proposed, the main goal of this algorithm is to converge to a Nash equilibrium in self-play without needing a Nash equilibrium as input. Such a multiagent learning algorithm can be used for two purposes:

- It can form the basis for problem specific algorithms.
- It forms an easy to implement alternative to the Nash equilibrium finders such as the Govindan Wilson method.

In this thesis it has been shown that the proposed algorithm is the first algorithm published to meet this goal. Other algorithms proposed either need a Nash equilibrium as input or do not converge in self-play to a Nash equilibrium in games with more then two players.

Compared to the exact Nash equilibrium finders this algorithm forms a good alternative. Although it is much slower then the exact solvers and it is not proven to converge. It is much easier to implement and does not need a deep mathematical understanding of the subject. In practise its performance measured in the number of games solved is comparable to the exact solvers, the exact solvers might be proven to be able to solve every game instance, but the reference implementations available get stuck on many games used in this test set and are therefore able to solve the same number of games as when our algorithm is used in self-play. The only advantage of the exact solvers is that they find all Nash equilibria and if they do not get stuck in an infinite loop or crash, they find the answer much faster.

6.1 Further work

The work on this algorithm is not complete, there is much which can be done to improve the algorithm. The extensions to this work can be put in three major categories: improving the algorithm as a multi-agent learner, improving its performance as a Nash-equilibrium finder and proving its claimed properties.

6.1.1 Optimal strategy finder

The current algorithm makes explicit use of the strategies of the other agents. To be able to use the algorithm in an online manner this should be changed to using only the observed actions of the other agents. This would increase the applicability of the algorithm.

It is well defined how the algorithm should handle when the other agent are learning using the same algorithm or have a stationary policy, it would however be very interesting if the algorithm was extended to give a good response against other multiagent learners when confronted with them. This subject is however hardly studied for any multiagent learning algorithm.

6.1.2 Nash equilibrium finder

For the algorithm to be used as an equilibrium finder its convergence properties should be improved, it should converges in all cases. One improvement which has been suggested already is to handle the situation in which one of the agents plays an almost pure strategy, this case is not a problem in theory since the algorithm does recover from it given enough iterations, but it is a problem in practise. This could be solved in several ways:

- Only use initial strategies with a high entropy.
- Use an alternative for the softmax function.
- Instead of using a gradient, switch to using Newtons method [30]. This will take the Hessian into account and will increase the steps at the edges where the derivative is getting infinitely small.

The first suggestion will not help in all cases, since the algorithm sometimes converges to a pure strategy if its initialized with a high entropy strategy, but it will help in the cases where the algorithm gets stuck after being initialized with a pure strategy. The second option could give a solution in all cases, it should however be a procedure that is still differentiable. A solution could be to adopt the softmax to make it less steeper in the middle and more steeper on the side.

The problem with the third suggestion is that the Hessian becomes almost non-invertible in the boundary regions, which causes the inverted Hessian matrix to be unstable.

The research on this algorithm can also be taken into a different direction. If the focus of the algorithm is purely on finding Nash equilibria and not on multiagent learning, a more global approach could be taken in which not every agent updates its strategy according to its own derivative, but a global minimization approach is taken. In such an approach the whole unconstrained vector denoting the joint strategy, σ, is updated according to $\frac{\partial G}{\partial \sigma}$. In order for G to be differentiable it should be redefined in terms of the softmax, replacing equation 4.1 and 4.2 with:

$$G_i = \sum_{a=0}^{M_i} \frac{e^{u_{ia}}}{\sum_{a'=0}^{M_i} e^{u_{ia'}}} (U_i - u_{ia}) \tag{6.1}$$

$$G = \sum_{i=0}^{N} \frac{e^{G_i}}{\sum_{j=0}^{N} e^{G_i}} G_i \tag{6.2}$$

If this definition is used $\frac{\partial G}{\partial \sigma}$ does exists and the joint strategy can be updated using a gradient descent. This could give a method purely aimed at finding a Nash equilibrium. It is however probable that the concept of Wolf has to be incorporated in some way in this method.

6.2 Theoretical Work

The current method is purely based on empirical observations, currently there is no proof that the method indeed converges to a Nash equilibrium in all settings, the construction of such a proof would be a giant step for the multiagent field. Smaller but still significant steps would be:

- An understanding of the Nash equilibrium the algorithm converges to, do all Nash equilibria have the same chance of being picked?
- Is it possible to give a better classification of the games for which the algorithm converges?

No attempt has been made during the research leading to this thesis to answer these questions.

6.3 Summary

This thesis contains several new ideas and insights:

- The concept of gain to determine the agent which is losing.
- Updating the strategy such that the probability of actions of which the expected reward is higher then the current expected are increased and the probability of actions of which the expected reward is lower then that of the current expected reward are decreased, is equal to updating all strategies according to $\frac{\partial U_i}{\sigma_i}$.
- Using the softmax rule to guarantee that all strategies are proper probability distributions.
- Solving the Nash equilibrium finding problem as a gain minimization problem.

These insights are combined into a working algorithm which has far better convergence properties then any algorithm proposed in literature. In the few cases that the algorithm does not converge the cause is well understood and several solutions for these causes are proposed. Overall it can be stated that the proposed algorithm takes the field a big step closer to the goal of building MAL algorithms that learn an optimal strategy against opponents of which the MAL-algorithm is known.

BIBLIOGRAPHY

[1] R. J. Aumann and M. Maschler. Game theoretic analysis of a bankruptcy problem from the talmud. *Journal of Economic Theory*, 36(2):195–213, 1985.

[2] Bikramjit Banerjee and Jing Peng. Performance bounded reinforcement learning in strategic interactions. In *Proceedings of the 19th National Conference on Artificial Intelligence.*

[3] Avrim Blum and Yishay Mansour. *Learning, Regret Minimization, and Equilibria*, chapter 4, pages 79–102. Cambridge University Press, 2007.

[4] H. Bosse, J. Byrka, and E. Markakis. New algorithms for approximate nash equilibria. In *3rd International Workshop On Internet And Network Economics*, 2007.

[5] Michael Bowling and Manuela Veloso. Convergence of gradient dynamics with a variable learning rate. In *Proc. 18th International Conf. on Machine Learning*, pages 27–34. Morgan Kaufmann, San Francisco, CA, 2001.

[6] Michael H. Bowling and Manuela M. Veloso. Rational and convergent learning in stochastic games. In *IJCAI*, pages 1021–1026, 2001.

[7] John S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. pages 211–217, 1990.

[8] G. Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 1951.

[9] V. Conitzer and T. Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents, 2003.

[10] Peter Cramton, Yoav Shoham, and Richard Steinberg. *Combinatorial Auctions.* The MIT Press, 2006.

[11] C. D'aspremont, Jaskold J. Gabszewicz, and J. F. Thisse. On hotelling's "stability in competition". *Econometrica*, 47(5):1145–1150, 1979.

[12] Drew Fudenberg and David K. Levine. *The Theory of Learning in Games.* MIT Press, 1998.

[13] S. Govindan and R Wilson. A global newton method to compute nash equilibria. *Journal of Economic Theory*, pages 65–86, 2003.

[14] Peter Hammerstein and Reinhard Selten. Game theory and evolutionary biology. In R.J. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 2, chapter 28, pages 929–993. Elsevier, 1994.

[15] Ehud Kalai and Ehud Lehrer. Rational learning leads to nash equilibrium. *Econometrica*, 61(5):1019 – 1045, September 1993.

[16] Simon Kirby. *Language Evolution (Studies in the Evolution of Language)*. Oxford University Press, USA, October 2003.

[17] E. Kohlberg. *Refinement of Nash Equilibrium: The Main Ideas.* Harvard Business School, 1989.

[18] H. W. Kuhn. Preface to waldegrave's comments: Excerpt from montmort's letter to nicholas bernoulli. *Precursors in Mathematical Economics: An Anthology*, pages 3–9, 1968.

[19] C. E. Lemke and Jr. J. T. Howson. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, june 1964.

[20] G. A. Malcolm. *The Princely Court: Medieval Courts and Culture in North-West Europe.* Oxford University Press, 2003.

[21] Richard D. McKelvey, Andrew M., McLennan, and Theodore L. Turocy. Gambit: Software tools for game theory, version 0.2007.01.30, 2007.

[22] Jorge J. Moré and David J. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Softw.*, 20(3):286–307, 1994.

[23] M. Muller. Computer go. *Artificial Intelligence*, 134:145–179, 2002.

[24] John Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences*, pages 48–49, 1950.

[25] Monty Newborn. *Kasparov versus deep blue: computer chess comes of age.* Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[26] E. Nudelman, J. Wortman, K. Leyton-Brown, and Y. Shoham. Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms, 2004.

[27] Martin J. Osborne and Ariel Rubinstein. *Bargaining and Markets (Economic Theory, Econometrics, and Mathematical Economics).* Academic Press, April 1990.

[28] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

[29] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a nash equilibrium. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 2004.

[30] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Newton-Raphson Method for Nonlinear Systems of Equations*, chapter 9.6, page 379. Cambridge University Press, 1992.

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

[32] Tuomas Sandholm, Andrew Gilpin, and Vincent Conitzer. Mixed-integer programming methods for finding nash equilibria. In *AAAI*, pages 495–501, 2005.

[33] Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artif. Intell.*, 171(7):365–377, 2007.

[34] B. Von Stengel. *Handbook of Game Theory with Economic Applications.* Elsevier, Amsterdam, 2002.

[35] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[36] G. van der Laan, A. J. J. Talman, and L. van der Heyden. Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. *Math. Oper. Res.*, 12(3):377–397, 1987.

[37] R. J. Vanderbei. *Linear Programming: Foundations and Extensions.* Kluwer Academic Publishers, Boston, 1996.

[38] J. von Neumann and O. Morgenstern. *Theory of games and economic behavior.* Princeton University Press, Princeton, 1944. Second edition in 1947, third in 1954.

[39] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[40] H. Young. *Strategic Learning and its Limits.* Oxford University Press, 2004.

[41] William R. Zame and John H. Nachbar. Non-computable strategies and discounted repeated games. *Economic Theory*, 8(1):103–122, 1996.

[42] Enst Zermelo. Uber eine anwendung der mengenlehre auf die theorie des schachspiel. In *Proceedings Fifth International Congress of Mathematicians*, volume 2, pages 501–504, 1913.

[43] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Twenty-First Annual Conference on Neural Information Processing Systems*, 2008.