Decision Making in Hostile Environments with Incomplete Information

- an application of reward-based learning for monetary decisions -

Bachelorarbeit

im Studiengang Wirtschaftsinformatik
der Fakultät Wirtschaftsinformatik
und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Verfasser: Thorsten Hermann Spieker
thorsten.spieker@gmail.com

Gutachterin: Prof. Dr. Ute Schmid, Universität Bamberg

Datum: 05. August 2010
Abstract

Decision-makers in businesses frequently have to make decisions based on the information that is available to them. Usually, this information is incomplete and sometimes a faulty representation of the real world. Management support systems can include semi or fully automated agents in order to augment this process. Modeling the input of those agents requires a suitable abstraction of the real world and proper algorithms in order to make the best decisions possible. Usually, the quality of a decision can be measured with a monetary value.

This thesis presents an automated approach to decision making using machine-learning algorithms for both abstracting and decision-making, based on previously observed events. This approach can be applied to any situation where decisions need to be made based on incomplete information and where the quality of the decision can be evaluated in a monetary way. This approach is however especially suitable to minimize short-term variations that may not represent the true expected effect of a decision. Such processes are the basis of all financial markets and also have countless applications in many aspects of businesses where managers need to make decisions that directly impact the profits of their company.
Table of Contents

List of Figures ........................................................................................................... V

1 Introduction ............................................................................................................. 1
  1.1 Business Decisions ......................................................................................... 1
  1.2 Texas Hold’em Poker ..................................................................................... 1
  1.3 Relevancy of Poker in Business Decision Context .................................... 2

2 Background ............................................................................................................ 3
  2.1 Relevant Fields of Research ........................................................................... 3
    2.1.1 Information Systems ................................................................................. 3
    2.1.2 Cognitive Systems ................................................................................... 3
    2.1.3 Intelligent Agents ................................................................................... 4
    2.1.4 Combining Information Systems and Cognitive Systems ................... 4
  2.2 Game Theory ................................................................................................. 5
    2.2.1 Extensive Games with Imperfect Information ........................................ 5
    2.2.2 Poker in Game Theory Context ............................................................ 6
  2.3 Reinforcement Learning ............................................................................... 6
    2.3.1 Reinforcement Learning in Games ....................................................... 7
    2.3.2 Reinforcement Learning in Poker ......................................................... 7
  2.4 Abstracting the Real World ........................................................................ 8
    2.4.1 Abstracting Texas Hold’em Poker ....................................................... 8
  2.5 Poker substitutes Business Environment .................................................. 9

3 Information Selection ......................................................................................... 10
  3.1 Handling Information Flow .......................................................................... 10
  3.2 Discarding Irrelevant Information .............................................................. 11

4 State Description ............................................................................................... 12
  4.1 Describing the Current State ...................................................................... 12
  4.2 Constructing a State Tree ............................................................................ 12

5 Reward Function ............................................................................................... 14
  5.1 Measuring the Outcome ............................................................................. 14
  5.2 Calculating Expected Outcome ................................................................. 14

6 Tradeoff Decisions ............................................................................................. 15
  6.1 Making a Decision ....................................................................................... 15
  6.2 Robust Decisions ......................................................................................... 16
  6.3 Exploitation vs. Exploration ...................................................................... 17
  6.4 Short-Term vs. Long-Term ......................................................................... 18
7 Results ........................................................................................................................................ 19
  7.1 Implementation Details ........................................................................................................ 19
  7.2 Competition Results ............................................................................................................ 19

8 Conclusion ................................................................................................................................ 21
  8.1 Future Work .......................................................................................................................... 21
  8.2 Acknowledgments .................................................................................................................. 22

Bibliography .................................................................................................................................. 23
List of Figures

Figure 1: Poker Server Protocol .................................................................................................. 11
Figure 2: PokerStars.com Sample Table .................................................................................... 11
Figure 3: Agent's performance in the total bankroll competition ............................................ 20
Figure 4: Agent's performance in the instant run-off competition .......................................... 20
1 Introduction

1.1 Business Decisions

In businesses managers need to make decisions that influence the whole company and set the path on which the company will walk. These decisions have to be made with the information available at the time. While it may be easy to make decisions where the amount of information required is small (for a manager that would have a low amount of responsibility), it becomes increasingly difficult as the responsibility and the amount of information increases. The bigger a company gets, the higher both this amounts go for a manager. On top of that, there are competitors - different companies in the same market - that try to undermine the actions of their rival, which adds another level of risk to making a decision. If one could handle all that, there is yet another issue: the available information is incomplete and may not represent the real world as accurately as needed. So what might look like a good decision - given the information - might turn out to have completely different and potentially bad effects in the real world. The outcome of a decision can usually be measured in a monetary value, like costs or profits.

Therefore, business decision-makers could use the help of automated systems, which select the relevant information from the flow of information and build a proper abstraction of the real-world from that information, giving predictions about which decisions will have which outcome.

1.2 Texas Hold'em Poker

The game of Texas Hold'em Poker is a card game where a player needs to make decisions and take actions based on the partial information that is known to him. He is playing to win money against other players that, in turn, try to win money from him as well. The information known to a player are his private hole cards (the two cards he has in front of him, which only him knows) and the public board cards (also known as community cards, these cards are shared by all the players). The missing information is the private hole cards held by each of his opponents. The outcome of a game of poker in the short-term is determined by a high degree of chance. However the long-term outcome is mostly determined by a high degree of skill or rather the strategy that is played. The money won or lost measures the outcome.
The rules in short are as follows:

Every player has a certain amount of chips or money in front of him with which he plays the game. Each player receives two private cards in the beginning of the game. Players now get four rounds of action in which they can fold (give up their cards, and forfeit any money they put in the pot so far), call (agree to a size of the wager), bet (setting the wager) or raise (increase the wager). Between the action rounds, five public board cards are dealt. After the final round of betting, all remaining players show their cards and the highest five-card combination out of the private cards of a player and the public board cards wins everything.

1.3 Relevancy of Poker in Business Decision Context

At first sight, poker might not seem suitable as an example for business decisions. However, as described in the previous two chapters, the decisions a player has to make in poker are very similar to the decisions a manager had to make for his business:

1. Both are acting in a hostile environment, where opponents want to get the best of them
2. Both need an abstraction of reality to make a decision, because enumerating all possible outcomes is impractical
3. Both are acting on the basis of incomplete information that may be misleading
4. Both actions have outcomes that are easily measured by a monetary value
5. Both outcomes may be influenced by higher power short-term but start to be representative long-term.

Therefore poker can serve as an excellent example to apply methods and concepts for automating business decisions without the need of an actual business and everything that comes with it, e.g. lack of cooperation, secrecy and risk.
2 Background

2.1 Relevant Fields of Research

2.1.1 Information Systems
The field of information systems combines the fields of computer science and business administration, to create a new field of studies needed to bring the best parts of both worlds together. It deals with the analysis of real world business processes, the abstraction of these processes into models and the aid of decision-making using, on the base of these models. The key concept is determining how to use available information and automate these processes as much as possible. The information gathering and the automation then help leading managers of a company to make proper business decisions (Ferstl/Sinz 2006, p. 1 and Grochla 1975).

Based on that, one can identify the key operations of a proper business analysis in terms of information systems:

1. Identification of the problem
2. Identification of the relevant real world
3. Identification of tasks and task handlers
4. Identification of information flow
5. Structural modeling of the process
6. Dynamic modeling of the process
7. Identification of automatable sub-processes
8. Implementation of automated sub-processes.

(derived from Ferstl/Sinz 2006, p. 91ff, 121ff, 443ff)

This ordered list of operations needed to properly develop a supporting information system will later serve as a reference. It is, however, by no means a complete list of available methods and operations in the field of information systems.

2.1.2 Cognitive Systems
The field of cognitive systems is a part of the computer science field which researches ways to create and maintain artificial intelligence. It tries to analyze humans and their behavior to identify structures, and rules that can be implemented in an artificial brain to automatically solve certain tasks (Russel/Norvig 2002, p. 3).

In either case, one needs to perform multiple tasks when designing and creating such an artificial brain:
1. Identification of the problem
2. Identification of the relevant real world
3. Modeling of the real world to serve as input
4. Selecting proper algorithms to solve the problem
5. Implementing the brain

(derived from Schmid 2009)

This list will - again - serve us later as a reference but is not intended to be a complete list of available methods and operations in the field of cognitive systems.

2.1.3 Intelligent Agents
When speaking of an artificial brain, there needs to be some clarification about what such a brain is. The definition of an intelligent agent would be an agent able to act - on its own – based on the information available to it. Such an agent is not only supposed to take every action required but also take reasonable actions based on an intelligent reasoning process.

“An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives” (Wooldridge 2009, p. 21).

2.1.4 Combining Information Systems and Cognitive Systems
As the attentive reader might already have noticed, the two aforementioned lists contain some overlaps. Therefore, it makes sense to say that the field of cognitive systems can help in developing information systems. It is possible to analyze and model a business process down to a structural abstraction of the real world, and a dynamic abstraction of the information flow to serve an intelligent agent as inputs. The input needs to model the environment where the agent is situated in and the agent needs to have a set of objectives to achieve. It then automatically solves the problem at hand by making decisions on its own or giving advice to human decision-makers.

Using the proper algorithms, such an agent could be trained to achieve different goals like maximization of profits, minimization of costs or picking the right market. It could also work with certain restrictions that need to be met like consistent number of employees or constant reduction of emissions. If the correct algorithms are selected, the agent can even improve over time by learning from past decisions and their monitored outcome.
2.2 Game Theory

Game Theory is a field that tries to explain the ways in which decision-makers interact. It assumes that the decision makers act rationally and take into account what actions their opponents might or might not take and for which reasons. Their behavior can therefore be described by objectives they want to achieve, which are based on the decision-makers experience and knowledge. The models obtained with this approach are highly abstract and can therefore be applied to a great number of real-life situations (Osborne/Rubinstein 1994, p. 1f).

A game in this context is a description of possible actions that players in the game can take but do not have to. It includes constraints on the actions for certain states of the game and therefore indicates which different states are possible and which are not. Usually, these states are defined by the sequence of actions taken by the players.

Four types of games exist: strategic games, extensive games with or without perfect information and coalitional games. These types are separated by different attributes:

- existent or nonexistent cooperation between the players,
- perfect or imperfect information and
- timing of decision (all players take an action at the same time or after each other).

2.2.1 Extensive Games with Imperfect Information

The model of an extensive game with imperfect information (also known as incomplete information) enables the possibility to let a player forget about previous actions. Either he forgets or he did not have the information in the first place. This describes a new category of games that applies to many real-world scenarios.

On the contrary to other game models where the player also does not have complete information, e.g. strategic games, the player gets to make a decision after certain actions have happened but that he is not aware of. In strategic games, all players decide at the same time which action they want to take which makes it obvious that players can not know about the opponents actions at the time of their decision (Osborne/Rubinstein 1994, p. 197ff).
2.2.2 Poker in Game Theory Context

A game of poker can be described as a set of actions taken by players (e.g. fold, call, raise), combined with chance events (cards dealt). A chance player represents the chance events and each possible card dealt is a possible action for the player. The strategy of the player consists of an equal distribution over all possible actions (cards).

Players do not cooperate; they are acting in a hostile environment. At the time of decision a player already knows about his opponent’s actions up to the point of decision. Therefore a player can have perfect information about the actions made by players but does not have any information about some of the actions of the chance player (e.g. cards dealt to opponents as private hole cards).

Which leads to the conclusion that poker is an extensive game with incomplete information.

2.3 Reinforcement Learning

Mitchell gives a pretty straightforward definition of reinforcement learning in his textbook on machine learning:

"Consider a learning robot. The robot, or agent, has a set of sensors to observe the state of its environment, and a set of actions to alter this state. [...] Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals. [...] We assume that the goals of the agent can be defined by a reward function that assigns a numerical value – an immediate payoff – to each distinct action the agent may take from each distinct state." (Mitchell 1997, p.367ff)

According to the specifications given in the introducing chapters regarding the automated agent, an artificial intelligence approach is needed which makes decisions based on a state that is constructed by gathered information. In addition, it is supposed to learn from past events of which it still may not have complete information about. Since the outcome of a decision is measured in a monetary value that is going to be maximized (or minimized) an expected reward can be calculated for each action in each state. As the quote above clearly shows, reinforcement learning is an exact fit for this purpose.

First, a state describing the current environment needs to be created. According to this state, possible actions that the agent can take can then be identified. For each of these actions, an expected reward can
be calculated. The agent can then take the action which will return the highest calculated reward. The Q learning algorithm has been developed for this exact purpose. It learns the reward function that determines a certain reward for taking an action \( a \) in a certain state \( s \):

**Algorithm 1: Q learning algorithm**

Parameter: A number of states \( s \)
Parameter: A number of actions \( a \)
Parameter: A discount factor \( y \)

1. For each \( s, a \) initialize the table entry \( Q(s, a) \) to zero.
2. Observe the current state \( s \)
3. Do forever:
4. Select an action \( a \) and execute it
5. Receive immediate reward \( r \)
6. Observe the new state \( s' \)
7. Update the table entry for \( Q(s, a) \) as follows:
   \[ Q(s, a) \leftarrow r + y \max Q(s', a') \text{ of all } a' \]
8. \( s \leftarrow s' \)

(Mitchell 1997, p. 375)

2.3.1 Reinforcement Learning in Games

Reinforcement Learning has been applied to approach a number of games already. Some of these games are:

- Go (moderate success)
- Chess (moderate success, good in endgame)
- Othello (weak success)
- Tic-Tac-Toe (a.k.a. “Noughts and Crosses”, great success)
- Connect-4 (a.k.a. “Four in a row”, great success)
- Backgammon (great success)
- Real-time Strategy Games (moderate success)

(Ghory 2004, p. 45ff, 51f)

2.3.2 Reinforcement Learning in Poker

Although reinforcement learning may seem to be the most appropriate fit as an artificial intelligence method to build an automated agent for poker, it has not been researched much. Most solution concepts aim to achieve a game theoretic optimal solution that is not exploitable without maximizing payoffs also known as an equilibrium strategy. But this is not how human players approach the game. They try to find the weaker players at the table and play a strategy that extracts as much money as possible from them what is called an exploitative strategy. For an agent to copy that style, one new aspect needs to be handled: opponent modeling. At the opposite, the beauty of optimal strategies is that it is
totally irrelevant whom the agent is playing against, so modeling the opponent is neither needed nor done.

The few papers that have been published on reinforcement learning in poker don’t handle the full game, but a smaller, similar game. They did however look promising. An agent that has been created in a 1-card version of poker performed well against a number of different artificial strategies (Wahab 2005, p. 7) and another agent created for the full heads-up game seems to have been doing even better against other strategies. Unfortunately, there is little more information available on this implementation than what’s given on the project’s wiki page (POSG 2007).

2.4 Abstracting the Real World

As part of the information systems studies, many approaches have been researched to abstract the real world of business processes and decisions. Common approaches are semantic object modeling (SOM), but you also find custom approaches that make use of the unified modeling language (UML). Both approaches take care of modeling both the structural and the dynamic view of a business process and deliver the models needed to create information systems and applications that automate (parts of) the process.

As the modeling of a business process is a whole topic on itself which is not a necessary part of this thesis, a further description of the topic will be skipped.

Instead a short description of current abstraction methods for the game of poker will follow, since the example environment for testing the approach of automated decision-making will be the game of poker as a representative for a business decision context.

2.4.1 Abstracting Texas Hold’em Poker

The problem with solving the game of poker is its complexity. With the number of possible combinations for a 7-card hand and the actions available, the search space becomes too big to store a representation of the full game. Therefore multiple abstraction methods are needed to reduce the search space to a manageable size, in order to actually be able to calculate a strategy for the game.
Those methods are:

1. Bucketing cards with strategic similarities
2. Reducing the number of raises allowed
3. Bucketing raises with similar raise amounts

This leads to an abstracted game in which a strategy can be found. If the abstraction was done so that key aspects of the original game remain intact, and identifiable in the abstract game, then a strategy found in the abstracted game can - but is not guaranteed - to be an equally successful strategy in the original game (Spieker 2010, p. 9ff and Johanson 2007, p. 22ff).

2.5 Poker substitutes Business Environment

As described in the previous chapters, the game of poker can act as a substitute in every aspect to a business environment, when it comes to being relevant to building an automated agent for decision-making. Therefore poker will serve as a representative from here on out.

Methods introduced for the different aspects will be presented as they apply to the game of poker, but they can also be seen as more general methods that apply to any scenario where a decision needs to be made in a hostile environment based on incomplete information. The outcome of the decision also has to be measurable in a monetary value so that reward functions can be defined.
3 Information Selection

3.1 Handling Information Flow

The first step in building our automated agent is handling the information flow. A poker game has a lot of information that needs to be taken into account. Since the agent is supposed to work on its own, a computer poker game representation is needed. This is possible in multiple ways:

1. Online Poker Client
2. Free Poker Server
3. Custom Poker Environment

The custom poker environment is the easiest to handle. One has total control over what the environment is doing so it can directly pass information to the competing players. However coding the whole environment is a duplicate work as there are poker servers available that handle the game flow already. So this only makes sense to build if and only if it really adds value above just being able to run agents against each other.

Free poker servers that are available to download and run, like the poker server that is used for running the Annual Computer Poker Competition (ACPC), may require coding an adapter that handles the information flow so our agent can understand it. That however is a more manageable task than coding an entire server as a custom environment.

Popular online poker sites offer games for humans only. This means they deliver their game in a graphical user interface that is not necessarily suitable for an automated agent. This means one needs to code an adapter that translates images into a game representation that the agent can handle. This is by far the most complex tasks of the three possibilities we’ve been considering, but it gives the ability to evaluate the agent against real human players and not just against other artificial computer opponents.
3.2 Discarding Irrelevant Information

When handling the information flow, it is necessary to filter available information in order to only consider the relevant ones. Depending on the complexity of the game provider (online client, server or custom environment), the amount of irrelevant information differs quite heavily. In a custom environment, one controls precisely which information is passed to the agents, which means no more work would be required.

When using a third party poker server, the information is already tailored to the usage of automated agents. Because such a third party server has been developed for the goal of running automated agent, the authors of such servers/frameworks already had boiled down the information to what truly is important. However some of that information may still not be necessary for making a decision. For example, the poker server used at ACPC sends a description of the currently ongoing game, called a game state, that includes the number of games played by both agents. This information usually should not have any effect on the decision of the agent and can therefore be discarded.

Information provided by an online poker client is modeled to represent the game of poker as entertainment; therefore it has multiple animations, pretty colors and appealing graphics. Players have names and avatars and there is also a chat where players can talk to each other. All that information should not have any effect on the agent’s decision and can therefore be discarded.

\[
\begin{align*}
\text{<matchstate> := ”MATCHSTATE,”} & \text{ <seat> := ”,”} \text{ <handNumber> := ”,”} \text{ <gamestate>} \\
\text{<handnumber> := <integer>} \\
\text{<seat> := ”0”|”1”}
\end{align*}
\]

Figure 1: Poker Server Protocol

Figure 2: PokerStars.com Sample Table
4 State Description

4.1 Describing the Current State

For the agent to make a decision, it needs to know in which state its environment currently is. Therefore, one needs to find a proper representation of the current state, as an input for the agent. There are multiple ways to describe state in a poker game.

As we've seen in the previous screenshot, online site use a graphical representation. Obviously these are tailored to human needs, so they can know precisely what's going on. For an automated agent however, they are practically useless. The information extracted from this representation should be used to build a new data structure, on which the agent can work. Two options are then available: one can define a completely custom data structure or simply reuse an API that has been published for this purpose.

The Meerkat API, published by BioTools Incorporated (Poker-Academy.com 2010), was originally created so that third parties could develop poker agents for their own proprietary poker software (Poker Academy Pro). Nowadays, it serves as an API for many custom poker agent designers. The benefit is that such agents can easily be integrated in different environments without much effort of having to adapt the information handler.

Another common representation in the academic world is the way the poker server for the ACPC describes a state in the game. Simply stated, it is a textual representation of the betting sequence. It does not handle game logic aspects, such as which player should act.

4.2 Constructing a State Tree

Another internal game description is possible for an agent, what can be called a game tree. Every instance of the game is represented by a node in the tree and every action, be it from a real player or the chance player, links the two states together. In poker, a state is represented by the cards each player gets and the action sequence that has happened so far. Since the cards of the opponent remain hidden until the very end of the game, one could say a state is only defined as the agent's cards, the board cards visible and the action sequence that led up to a certain point in time in the game. When trying to build the full tree however, it
becomes evident that the full game is too complex to be stored in a tree. Which means it needs to be abstracted, as we previously described briefly in 2.4. The node that represents the current state is then passed to the agent who makes a decision based on that information. The actions available can be extracted by enumerating the children of the node.

For this experiment, an action sequence tree is created as defined in a previous thesis (Spieker 2010, p. 10). Each node in the tree has a number of attributes that describe the state. This includes the size of the pot (all chips in the middle) the number of times the state has been observed and a list of shown down cards at the end of the game. The shown down cards are from opponents and can therefore be used to approximate likely holdings of an opponent. The number of times a state has been observed is useful to calculate the probability of a state happening, e.g. a current node (or state) has been observed 100 times, it's children have been observed 20, 30 and 50 times. Then the probability of each of the child states happening is 20%, 30% and 50% respectively.
5  

5.1 Measuring the Outcome

In poker, the outcome of a decision is immediately observable. Next to the observed new game state after we take an action, the agent also has less money available for actions from that point on. The payout however does not come until the end of the game. This means either the agent folds, the opponent folds or there is a showdown (last betting round is over and players show their cards).

If the agent folds, it loses everything it has committed to the pot. If the opponent folds the agent wins everything the opponent has committed to the pot and gets its money back. If there is a showdown three different possibilities arise:

1. The agent's cards have a higher value than the opponents cards
2. The agent's cards have a lower value than the opponents cards
3. Both players have cards with equal value.

In the first case the agent wins everything and in the second case, the opponent wins everything. In the last case, the pot is split between the players. In order to measure the outcome, one only needs to take the difference between the chips in front of a player at the beginning of the game and after the game. The change in the number of chips is then the outcome of the game: a win, a loss or no change.

5.2 Calculating Expected Outcome

To calculate the expected outcome at a certain node, one needs to consider the current holding of the agent and the information available about the current game state, supplied within attributes of the game tree node. One can then define an algorithm that calculates the expected outcome in a certain situation such as this one:

**Algorithm 2:** getExpectedOutcome(s, c)

Parameter: The state s
Parameter: The agent's cards c

1. If s is a showdown node
2. Count wins, draws and losses of c against observed cards
3. Let p be the size of the pot in s
4. Return \((\text{wins}+\text{draws}/2)\cdot p - p/2\)
5. Else
6. Let e be 0
7. Loop over all children of s
8. Let s' be the child of s
9. \(e \leftarrow e + \text{getExpectedOutcome}(s', c) \cdot \text{probability of } s'\)
10. Return e
11. Endif
6 Tradeoff Decisions

6.1 Making a Decision

So far, we’ve covered the first few steps required by reinforcement learning. Firstly we’ve shown how a state can be represented, and how it can be differentiated from other states. Secondly, a method has been introduced that calculates the reward for any given state.

To make this compliant with our goals, we need to define two additional steps: what is the expected reward for taking an action in a certain state, and how does the agent learn from previous actions. When those two aspects have been covered, a policy can then be defined for the agent to always choose the action with the maximal expected reward, and our agent will be complete, as in able to play a game on its own (against other agents or humans).

As mentioned in 4.2, the agent can enumerate all children of a certain node in the tree in order to get actions available at this point in time. Now, the question is which action returns which reward. For that, one can just use the expected outcome algorithm to calculate the expected outcome of the child that represents a certain action. This will give us the expected reward for that action. Now, according to our policy to always choose the highest expected reward, the agent only needs to compare the different expected rewards for all the actions and choose the most profitable one – the one with the highest expected reward.

The algorithm however relies on proper values for the occurrences of a node (which is needed to get the probability) and the observed shown down cards of the opponent. Only a proper learning algorithm can meet this condition. There are two ways to do the learning, offline while not playing the game, or online while playing. Reinforcement learning usually means online learning so this will be handled first.

When the agent plays the game there is an action sequence tree in the background that represents the entire game. When it is the agent’s turn to act, the node representing the current state is passed to the agent, which then makes a decision based on that information. As previously said, we calculated the expected rewards of the possible actions and chooses the one with the highest reward. When observing a certain state, either the agent or the outlying framework handling the action sequence tree increases the counter for the occurrences in the current node for future use. This way, when walking the tree from one state to
the next, the occurrences will be updated according to observed events. Doing this for a large number of games increases the accuracy of our nodes, so they can converge to the true probability of such a game state happening.

When observing a showdown, the shown down cards are just as easily storable in the respective showdown node. Since the only information about the opponent’s cards (due to the partial information available) is the action sequence, it is necessary to have an accurate model of which cards the opponent could be holding after a certain sequence of actions. By increasing the number of observed showdowns, the more representative the list of observed cards will become. It will converge to the actual likelihood of card holdings.

6.2 Robust Decisions

Therefore, accuracy of the expected reward algorithm is directly dependent on the number of observed games. Of course the agent can just play a huge number games and update the tree accordingly, but that would probably cost the agent a lot of money in the beginning of it’s play session. Instead, it is possible to give the agent a kick-start before actually letting it play by using offline learning.

As mentioned in a previous thesis, there is a database of poker games that contains over 600 million different games (PokerFTP.com 2009). One can use this database in order to populate the action sequence tree in an offline learning process. The nodes are then updated as the tree is walked through according to the observed game states. Assuming the database contains an accurate representation of common play, this offline learning process should definitely be valuable for the agent. And as previously stated, the accuracy increases with the number of observed games. We can safely assume that over 600 million games, we should manage to get an accurate representation that isn’t biased by any external factors.

This brings not only the ability to have a kick-start in a new competition, but also enable the agent to have a more versatile view of the strategies that are used by players in general. Had we only used information against one opponent, our reinforcement learning algorithm would have tailored to that particular opponent, leaving us open to being completely exploitable to a new strategy, used for example by a new player.
As the database consists of hands collected from a broad spectrum of players, the action sequence tree will represent a mixed strategy of play that represents all of these playing styles. Playing well against this opponent model should lead to a strategy that plays well against a big number of players and their strategies. While it might lose against a certain opponent in the beginning, it will perform well against most players and then adapt to the specific opponent it currently plays against to maximize profits.

6.3 Exploitation vs. Exploration

When using a policy that only takes the action with the highest expected reward, the agent could start getting hung up with one particular line of actions without exploring any other options. Also, when letting the agent play such a strategy, it should be fairly easy for an opponent to model the agent’s strategy accurately, and exploit those tendencies heavily to make the agent lose. With a big number of already observed games, the agent would not learn to adapt fast enough against such an opponent, as a rather small number of games do not have enough influence (weight on the tree) on the overall strategy. Therefore it is recommendable to let the agent mix its strategy from time to time, and also include barely negative expected rewards in that mix.

Instead of taking the action with the highest expected reward, the agent could collect all actions that have positive expected rewards and return one of these actions with equal probability. Or it could return an action with weighted probability according to the size of the expected reward. Instead of using only actions with positive expected reward, the agent could also start exploring actions with slightly negative expected rewards with a fixed probability, e.g. 90% of the time take actions with positive expected rewards and 10% of the time take an action with slightly negative expected reward.

To sum this up, it is recommended to mix its strategy when playing a good opponent. By being too obvious, we leave the door open for a good opponent to adapt his strategy to our own, and start exploiting our weakness. When playing weaker opponents that will not make any effort in trying to exploit our strategy, it is recommended however to stick to the precomputed model (Johanson 2007, p. 71).
6.4 Short-Term vs. Long-Term

When playing poker, it is essential to differentiate between short-term outcomes and long-term outcomes. Short-term, an outcome is heavily influenced by chance as the card distribution is dependent on chance. This means a negative outcome of a game could lead to the wrong conclusion that a certain choice of actions was bad or vice versa a positive outcome could lead to the wrong conclusion that a certain choice of actions was good. To counter this effect, it is essential to play a number of games first before deciding whether a certain strategy is good or bad.

Now, it might seem wrong to make the agent update the tree after each game. However with an increasing number of played games, the effect of each individual game on the overall sum of games that form the strategy becomes less and less. With that in mind, it is even more necessary to have the agent trained on a big number of hands offline before making it compete against opponents. As the number of observed games increases, the agent ability to adapt to a specific strategy decreases. This calls for a tradeoff between the number of observed games used for offline training to counter short-term effects and the update rate of the tree online to adapt to the opponent.
7 Results

7.1 Implementation Details

The introduced methods were implemented in Java, on top of already existing frameworks. The agent was primarily built against the Meerkat API and later extended to work with the poker server of the ACPC 2010. The frameworks for creating the action sequence tree and for using the PokerFTP.com database were taken from earlier work on abstracting poker. A custom testing framework was developed previously in cooperation with members of the PokerAI.org Forums (PokerAI.org 2009).

The action sequence tree can be created for a number of different abstractions, which means that after evaluating the methods to calculate a strategy, those abstractions could then be tested to have a look at which one return the best results in the full game. After creation, the tree can be passed to an offline learning process, which makes use of the game database. With or without that process, it is then serialized and persisted on the hard drive.

When starting the agent in either the testing framework or the ACPC framework, it first loads the tree from disk to memory. When then presented with a new state, it will act according to the previously defined policy, either by taking the action with the highest expected reward or any of the actions with positive expected reward. The agent internally expects Meerkat GameInfo objects as the state representation and also returns Meerkat Action objects as actions. Therefore, an adapter had to be implemented for parsing the ACPC game state description sent by their server and wrapping it in a GameInfo object. The returning Action objects are then converted into the ACPC compliant string representation, and sent back to the server.

7.2 Competition Results

The implemented agent was submitted to compete in this year’s Annual Computer Poker Competition. Up until now there are only preliminary results, but they already indicate a pretty strong tendency.

The agent “c4tw” was submitted in both no limit heads up competitions with differently trained versions. As an abstraction, a custom betting structure with four allowed raises per round and 6 allowed raises in total was used for both competitions. For the total bankroll competition the agent “c4tw.tbr” was trained over one million games prior to the compe-
tition while the agent “c4tw.iro” submitted for the instant run-off was not trained at all. It was assumed, that the untrained version would adapt much better to the opponent but would also be more exploitable while the trained version would not adapt as fast but would be much less exploitable due to its prior training.

<table>
<thead>
<tr>
<th></th>
<th>Round 0</th>
<th>Round 1</th>
<th>Round 2</th>
<th>Round 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tartanian4.tbr</td>
<td>2156 ± 48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PokerBotSLO</td>
<td>1458 ± 184</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperborean.tbr</td>
<td>1212 ± 26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SartreNL</td>
<td>537 ± 34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c4tw.tbr</td>
<td>-5362 ± 201</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Agent's performance in the total bankroll competition

<table>
<thead>
<tr>
<th></th>
<th>Round 0</th>
<th>Round 1</th>
<th>Round 2</th>
<th>Round 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperborean.iro</td>
<td>1351 ± 44</td>
<td>408 ± 27</td>
<td>224 ± 31</td>
<td>200 ± 39</td>
</tr>
<tr>
<td>SartreNL</td>
<td>581 ± 34</td>
<td>12 ± 23</td>
<td>-79 ± 27</td>
<td>-200 ± 39</td>
</tr>
<tr>
<td>Tartanian4.iro</td>
<td>1338 ± 33</td>
<td>-60 ± 27</td>
<td>-145 ± 34</td>
<td></td>
</tr>
<tr>
<td>PokerBotSLO</td>
<td>1620 ± 187</td>
<td>-359 ± 32</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>c4tw.iro</td>
<td>-4891 ± 213</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Figure 4: Agent's performance in the instant run-off competition

These results were pretty surprising and lead to multiple assumptions. First the implementation could still have bugs that have not been found yet. As a matter of fact the implementation of the agent was pretty rushed to meet the deadline of the competition and later bug fixes only fixed bugs in the translation of the ACPC game state to the Meerkat API and back.

In later investigations it even became evident that the implementation of the reward calculation includes a major error. This unfortunately renders the results pretty useless, especially against a competition which has competed for multiple years already and ironed out such issues.
8 Conclusion

8.1 Future Work

After presenting the methods and submitting a first implementation of an agent using these methods, there remains a good deal of possible applications for future work.

First, the implementation of the agent should be made modular so that reward functions and action policies can more easily be substituted without having to build the code again. This will tremendously help the testing process of different strategies, as it would remove the need to recompile the whole project for testing a new strategy.

Second, the function for calculating expected reward has to be revised in the implementation to correct implementation errors. This has already been done but unfortunately was too late for resubmitting the agent to the Annual Computer Poker Competition 2010. One practical detail learned from that experience is that a node should contain an additional attribute which indicates which player made this action. This will help future reward functions, and also the agent.

Third, an agent should act on different action sequence trees for each position he can sit at the table. This would stop mixing of opponent and agent’s past actions in one model and return a much clearer view of the opponent and its weaknesses.

Fourth, different reward functions and policies should be tested against multiple opponents to find the best among them. When testing different setups, different abstractions should also be tested to find out which one performs best. Note however that the quality of an abstraction can already be measured differently (Spieker 2010, p. 21).
8.2 Acknowledgments

This work would not have been possible without the help and support from a couple of people.

- First I would like to thank Prof. Dr. Schmid for supporting my research in poker and making it possible to write two theses about it. I certainly could not have had a better supervisor for this thesis.
- Next is Prof. Dr. Sinz for approving the title and abstract of this thesis. It did take a couple of tries before it was satisfactory and appropriate for my studies of information systems but in the end it all worked out.
- While working on computer poker in general I have gotten to know and learned to respect multiple members of PokerAi.org and I would like to thank every single one of them for collaborating and getting me into the topic. Special thanks go out to my proofreader among them. You will know it if you are either of them.
- Another great helper was Nolan Bard in his function as the chair of the Annual Computer Poker Competition 2010 who did not only find quite a few bugs in my code but also let me fix those after the given deadline.
- Most importantly my parents and my brother deserve special thanks for all the support they have given me during the time of my studies. Without them I certainly would have gone crazy over the load of work that I had put on myself.
- Finally, I would like to thank my friends at the University of Bamberg that made this an incredible ride and that have given my a fun three years of studies!
Bibliography


Ghory, Imran: „Reinforcement learning in board games.“ 2004, Department of Computer Science, University of Bristol

Grochla, Erwin: „Betriebliche Planung und Informationssysteme“ 1975, Verlag Rowohlt


Schmid, Ute: „Intelligente Agenten“ Slides for lecture 2009, Cognitive Systems Group, Otto-Friedrich Universität Bamberg


Erklärung

Ich erkläre hiermit gemäß § 17 Abs. 2 APO, dass ich die vorstehende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Bamberg, den 05. August 2010

____________________
(Unterschrift)