It's a magical world, Hobbes, ol' buddy... Let's go exploring!

– Calvin (Calvin and Hobbes).

Games lubricate the body and the mind.

– Benjamin Franklin.

Rock is an equilibrium.

– Michael Bowling.

University of Alberta

State Translation in No-Limit Poker

by

David Paul Schnizlein

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

©David Paul Schnizlein
Fall 2009
Edmonton, Alberta

Permission is hereby granted to the University of Alberta Libraries to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only. Where the thesis is converted to, or otherwise made available in digital form, the University of Alberta will advise potential users of the thesis of these terms.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Examining Committee

Michael Bowling, Computing Science

Duane Szafron, Computing Science

Jonathan Schaeffer, Computing Science

Bora Kolfal, Business

To my parents,

who made me who I am today.

Abstract

One way to create a champion level poker agent is to compute a Nash Equilibrium in an abstract version of the poker game. The resulting strategy is then used to play in the full game. With this approach, translation is required between the full and abstract games in order to use the abstract strategy. In limit poker this translation step is defined when the abstraction is chosen. However, when considering no-limit poker the translation process becomes more complicated. We formally describe the process of translation and investigate its consequences. We examine how the current method, hard translation, can result in exploitable agents and introduce a new probabilistic method, soft translation, that produces more robust players. We also investigate how switching between strategies with different underlying abstractions affects the performance of an agent.

Acknowledgements

First of all, I would like to thank **Maria Gini** for her guidance during my time at the University of Minnesota. She helped me get started performing independent research, which led directly to my work on poker.

Next, my supervisors **Duane Szafron** and **Michael Bowling** provided great support throughout my Master's work. Their different work styles meant that I always felt like I had someone who understood my situation. I could not have asked for better supervisors.

While still taking classes, one of the projects I worked on ended up as a section in my thesis. Great thanks goes to **Nick Abou Risk** and **Adam Harrison** for their work on the project that become Section 4.5.

When first starting my work with the Computer Poker Research Group, **Neil Burch** was an invaluable resource. His help learning the code base as well as his ceaseless criticism of my programming practices increased the quality of the work I performed.

I would also like to acknowledge the remaining members of the CPRG: **Morgan Kan**, **Josh Davidson**, **Nolan Bard**, **John Hawkin**, **Mike Johanson**, **Kevin Waugh**, **Jonathan Schaeffer**, **Bryce Paradis**, and **Richard Gibson**. Whether discussing game theory or playing poker late into the night, this group has been incredible to work with.

A big thanks goes out to **Carsten Moldenhauer** and **Rick Valenzano** for keeping me sane these two years. My time would not have been the same without our late night games of Skat and various holiday celebrations.

To everyone in the Games Research Group as well as the rest of the Computing Science Department at the University of Alberta, I would like to say that I truly obtained the best education I could have here. The people I met and worked with have been amazing, and by far I will most miss the ability to interact with such a wonderful group of people.

Finally, I would not be where I am today without my parents **Brian** and **Barb** and my brother **Rob**. Their constant support allowed me to achieve greater things, and I am looking forward to seeing them more often.

Table of Contents

1 Introduction **1**
1.1 Poker 2
1.2 Motivation 3
1.3 Contribution 5

2 Background **7**
2.1 Extensive Form Games 7
2.1.1 Definition 7
2.1.2 Poker Example 9
2.1.3 Best Response and Nash Equilibria 10
2.2 Equilibrium Algorithms 10
2.2.1 Linear Programming 11
2.2.2 Gradient-Based Algorithms for Finding Nash Equilibria 11
2.2.3 Counterfactual Regret Minimization 12
2.3 Abstraction 12
2.3.1 Definitions 13
2.3.2 Card Abstraction 14
2.3.3 Action Abstraction 16
2.4 Evaluation 17
2.4.1 Direct Play 17
2.4.2 DIVAT 19
2.4.3 Importance Sampling 20

3 Analysis of the No-limit Polaris Agent **22**
3.1 Introduction 22
3.2 Abstraction Allocation 22
3.2.1 Betting Abstraction 23
3.2.2 Card Abstraction 24
3.3 Polaris Translation 25
3.3.1 Exploitative Techniques 26
3.3.2 Translation Fixes 27
3.3.3 Results 28
3.4 Translation Concepts 29
3.5 Conclusion 31

4 State Translation **32**
4.1 Introduction 32
4.2 Hard Translation 33
4.2.1 Definitions 33
4.2.2 Weaknesses 35
4.3 Soft Translation 37
4.3.1 Definitions 37
4.3.2 Effects 38
4.3.3 Dominated Actions 39
4.4 Application to Poker 40
4.4.1 Hard Translation 40
4.4.2 Soft Translation 41
4.5 Boundary Exploration 41
4.5.1 Problem Definition 41
4.5.2 Maximum Likelihood Estimation 42
4.5.3 Estimation Algorithm 43

4.5.4 Results . . . 43
4.6 Results . . . 45
4.6.1 Opponents . . . 46
4.6.2 Abstract Translation . . . 46
4.6.3 Real Translation . . . 49
4.7 Conclusion . . . 50

5 Strategy Switching **51**
5.1 Introduction . . . 51
5.2 Perfect Information Games . . . 52
5.3 Imperfect Information Games . . . 52
5.4 Cover Set . . . 53
5.5 Results . . . 54
5.6 Conclusion . . . 56

6 Conclusion **57**
6.1 AAAI No-Limit Competition Results . . . 58
6.1.1 2008 Competition . . . 58
6.1.2 2009 Competition . . . 58
6.2 Future Work . . . 59
6.2.1 No-Limit Man-Machine Match . . . 59
6.2.2 Imperfect Recall Betting . . . 60
6.2.3 Translation as Transfer Learning . . . 61

Bibliography **62**

A Switching Data **64**

List of Tables

1.1 Performance results of the no-limit aspect of the 2007 AAAI Computer Poker Competition in smallbets/hand (sb/h) . . . 4

2.1 Money variance analysis of the 2008 no-limit competition in sb/h . . . 18
2.2 DIVAT variance analysis of the 2008 no-limit competition in sb/h . . . 19
2.3 Importance sampling effect on standard deviation in sb/h . . . 21

3.1 Card/betting abstraction trade-off performance of several 100BB agents in mb/h . . . 23
3.2 Card/betting abstraction trade-off performance of several 200BB agents in mb/h . . . 24
3.3 Imperfect recall performance of several 500BB fcpta agents in mb/h . . . 25
3.4 Translation fixes effects in mb/h . . . 28
3.5 Performance of real translation versus abstract translation in mb/h . . . 31

4.1 Effectiveness of EstimateBoundary at predicting a Polaris agent's boundary point . 45
4.2 Translation results for 500BB players in mb/h using abstract translation . . . 47
4.3 Translation results for 100BB players in mb/h using abstract translation . . . 48
4.4 Exploitability of various 12-stack Leduc Hold'em players in sb/h using abstract translation . . . 48
4.5 Translation results for 500BB players in mb/h using real translation . . . 49
4.6 Translation results for 100BB players in mb/h using real translation . . . 49
4.7 Exploitability of various 12-stack Leduc Hold'em players in mb/h using real translation 50

5.1 Exploitability of various cover sets in Leduc Hold'em in sb/h . . . 55
5.2 Switching results for half pot bets in the 200BB stack game in mb/h . . . 55

6.1 Performance results of the no-limit aspect of the 2008 Computer Poker Competition in sb/h . . . 58
6.2 Performance results of the no-limit aspect of the 2009 Computer Poker Competition in sb/h . . . 59

A.1 Full cover set for 12-stack Leduc Hold'em . . . 67
A.2 Sub cover set for 12-stack Leduc Hold'em . . . 68

List of Figures

1.1 Full game strategy development process 5

List of Symbols

Betting options:

- f - fold
- c - check/call
- p - pot bet
- a - all-in bet
- h - half pot bet (0.5*p)
- q - three-quarter pot bet (0.75*p)
- w - one and half pot bet (1.5*p)
- d - double pot bet (2*p)
- t - ten pot bet (10*p)
- e - eleven pot bet (11*p)

Abbreviations:

- BB - big blind
- sb - small bet (equal to one BB)
- mb - millibet (equal to 0.001 sb)
- h - hand, as in one hand of play

Approximate standard deviations for matches:

- 100BB: 5-10 mb/h
- 200BB: 10-20 mb/h
- 500BB: 15-25 mb/h

Chapter 1

Introduction

Games have always been a prevalent part of society. Going back as far as recorded history allows, we observe that basically every society played games of some type. Games stimulate the mind and provide a competitive environment and social outlet where people can strategize. In modern times, the best game players are world renowned for mastering games like backgammon and chess. With the rise in computing power and further development of game theory, it is not surprising that computers have become key players in some of these games.

Games have been used as a testbed for artificial intelligence research since John von Neumann and John Nash laid the foundation for game theory over 50 years ago [21, 18]. Since then game theory, heuristic search, monte carlo simulation and linear optimization have seen intense study and development. With the advent of current technological advances, much of this theory has been implemented in computers and resulted in champion level agents in chess (Deep Blue [11]), checkers (Chinook [26]) and even poker (Polaris [13]). There are several reasons why games have been used so much in artificial intelligence research.

Well Defined Rules: The rules in games are well defined. It is clear what the possible situations are and what actions are legal in any given situation. Every player is forced to play within the rules, making it impossible to play unexpected tricks.[1] This makes it easier to create agents to play games.

Clear Rewards: The possible rewards in games are clearly defined. Certain situations in a game have rewards associated with them. These rewards may be very simple, for instance in chess or checkers the reward is simply +1 for being in a winning state, -1 in a losing state and 0 elsewhere. In a game like poker, however, the reward represents the number of chips won/lost during the hand.

Easy Evaluation: Given the ruleset and reward functions, it is easy to evaluate a strategy or set of strategies for a given game. If the game is small enough, this can be a direct calculation using the given strategies. In other situations, the agents can be repeatedly played against each other to determine which is better.

[1] Though of course, a player may play in an unexpected manner within the ruleset.

1.1 Poker

The game of poker is used as the primary testbed for the work in this dissertation. Poker has been studied for some time [3] and has grown in popularity in recent years. In addition, three annual AAAI Computer Poker Competitions [33] have helped spur poker research.

Poker exhibits many properties that make it an interesting game to study. First, it is a very large game, which means that it is computationally infeasible to store the entire game in memory. Second, there is a large amount of stochasticity in the game in the dealing of cards. This means that luck is a factor in determining who wins an individual hand and makes it more difficult to differentiate which player has more skill. Third, poker is a game of imperfect information in that a player's cards are not visible to the other players. This means that techniques used for games like checkers and chess, games of perfect information, are in general not effective for poker. Finally, poker has many reward values instead of the standard 1 for winning and -1 for losing. In poker, the reward is the number of chips won/lost in a hand, and one of the main skills is the ability to manipulate this amount to win more chips and lose fewer chips.

Two variants of poker are primarily used in this dissertation. The first is Texas Hold'em. This game is the most popular variant of poker being played today and is most often the variant of poker shown on television. Additionally, this is the variant used in the annual AAAI poker competitions. The second variant is Leduc Hold'em. Leduc Hold'em is a much smaller game used for experiments and is not generally played by humans. This variant is used because it is small enough to theoretically solve, and thus we can obtain more accurate measurements when creating agents for this game.

Texas Hold'em is a game played with a standard 52 card deck consisting of 4 suits and 13 ranks. Each player makes the best 5-card poker hand possible according to the standard poker ranking. The game consists of several stages.

Blinds: One player is designated the dealer (this generally rotates after each hand). The player to the left of the dealer is the *small blind* who makes a forced bet whose size is half that of the *big blind*. The player to the left of the small blind is the *big blind* who makes a forced bet whose size is a *small bet*. A *small bet* is the unit amount determining the stakes of the game.

Pre-flop: All players are dealt two private cards only they can use, followed by a betting round.

Flop: Three community cards are revealed that all players can use, followed by a betting round. These community cards, and all subsequent community cards, are often referred to as the *board* cards.

Turn: One community card is revealed, followed by a betting round.

River: A final community card is revealed, followed by a final betting round.

Showdown: If more than one player has not folded during the betting rounds, the player with the best 5-card poker hand wins the money placed into the pot during the betting rounds.

The betting round differs slightly depending on the variant being played. In all variants, the

pre-flop betting round begins with the person left of the big blind, and all other betting rounds begin with the small blind. Every player can choose to either *fold* (forfeit their hand), *check/call* (match the largest current bet), or *bet/raise* (add additional chips to the pot that others must match). In *limit* Texas Hold'em, the amount raised is determined by the round and not by the players. In the first two rounds the bet size is one small bet, and in the last two rounds it is two small bets (one *big bet*). *No-limit* differs in that players can bet/raise any number of chips between a minimum bet and all of their chips. The minimum bet/raise is either the size of the bet currently faced or one small bet, whichever is larger.

Heads-up No-limit Texas Hold'em refers to the no-limit betting version of Texas Hold'em in which there are only two players. In no-limit, every player starts with a number of chips, referred to as their *stack*. An arbitrary number of no-limit variants can be created by varying the stack sizes of the players in the game. Throughout this dissertation three such games will be referenced, which have stack sizes of 500 big blinds (*BB*), 200BB and 100BB. The 500BB game was used in the 2007 and 2008 No-limit AAAI Computer Poker Competitions, whereas the 200BB game was used in the 2009 competition. The 100BB game is the stack size most commonly played by humans, and thus some experiments were run in this game as well.

In no-limit Hold'em a *small bet* (*sb*) is the size of the big blind as well as the minimum bet size when making a bet. In all of our no-limit variants the small bet is 2 chips (and thus the small blind is 1 chip). When evaluating players, we will generally refer to their performance in terms of small bets won per hand, or sb/h. The term *millibets* (*mb*) is also used, with one small bet equal to $1,000$ mb. All results will be given in either sb/h or mb/h, depending on the magnitude of the data.

Leduc Hold'em is a game similar to Texas Hold'em but much smaller. The Leduc game only has 6 cards: 2 suits with 3 ranks. Each player is dealt one private card, and the flop consists of only one public card. There are only two betting rounds, one after the private cards are dealt and one after the flop is dealt. At the showdown, there are no flushes or straights so the highest hand is a pair, followed by whoever has the highest cards.

The variant we use has stack sizes of 12 chips and each player antes 1 chip at the start of each hand. Since this game is so small, we can directly calculate the best response to any strategy. This means that given any strategy, we can compute a value that tells us how exploitable that strategy is. In Leduc Hold'em sb/h and mb/h are again used for player evaluation. However, since each player antes 1 chip, the small bet is equal to one chip instead of two.

1.2 Motivation

The research performed for this dissertation was inspired by a single point of data from the 2007 AAAI Computer Poker Competition [33]. In this competition I was partially responsible for creating a variety of agents, dubbed PokeMinn. These agents worked by performing expected-value computations on a model of the game. The two *limit* versions of PokeMinn had been well tested and

had their parameters tweaked to provide slightly different but good behaviors. The *no-limit* version of PokeMinn may or may not have been written the night before the competition deadline, fulfilling only the requirements that it run without crashing and within the time allotted. Needless to say, I was not expecting much to come out of this agent. However, the results of the competition, shown in Table 1.1, highlight an interesting event. In the table, every row denotes a player as *(X) Name*, with the column denoted by the same number corresponding to the same player. The intersection of two players shows the amount that the row player beat the column player by. Many similar tables will be shown in this dissertation. It is important to note that a player that folds every hand loses at most 0.75 sb/h.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	Avg
(1) BluffBot20		0.17	0.24	0.58	2.09	2.89	3.44	0.48	1.85	2.47	1.58
(2) GS3	-0.17		-0.08	0.50	3.16	0.12	1.88	4.20	-42.06	5.02	-3.05
(3) Hyperborean07	-0.24	0.08		-0.05	6.66	5.46	6.80	8.70	14.05	22.12	7.06
(4) SlideRule	-0.58	-0.50	0.05		11.60	9.73	10.34	10.39	15.64	10.79	7.49
(5) Gomel1	-2.09	-3.16	-6.66	-11.60		3.18	8.37	11.45	62.39	52.33	12.69
(6) Gomel2	-2.89	-0.12	-5.46	-9.73	-3.18		15.08	11.91	58.99	40.26	11.65
(7) Milano	-3.44	-1.88	-6.80	-10.34	-8.37	-15.08		5.74	12.72	27.04	-0.04
(8) Manitoba1	-0.48	-4.20	-8.70	-10.39	-11.45	-11.91	-5.74		18.82	50.68	1.85
(9) PokeMinn	-1.85	42.06	-14.05	-15.64	-62.39	-58.99	-12.72	-18.82		34.30	-12.01
(10) Manitoba2	-2.47	-5.02	-22.12	-10.79	-52.33	-40.26	-27.04	-50.68	-34.30		-27.22

Table 1.1: Performance results of the no-limit aspect of the 2007 AAAI Computer Poker Competition in smallbets/hand (sb/h)

Looking at Table 1.1, one may notice that the match between GS3 and PokeMinn appears to be an outlier. GS3 finished in second place, losing to only two other agents by a small amount. PokeMinn finished in second to last place, beating only one other agent. Regardless, PokeMinn beat GS3 by 42.06 sb/h, the largest amount (in absolute value) that any of the top 4 bots obtained against any opponent. This loss was so drastic it caused GS3's average performance to be -3.05 sb/h, the third lowest overall average. The competition was run in a bankroll runoff fashion. For this metric all players are considered, and the player with the lowest bankroll is removed. This process is repeated without the removed player(s) until only one player remains. Thus, when PokeMinn was eliminated GS3's average bankroll rose significantly. Obviously, this result was very strange and required further investigation.

After speaking with some of the other participants, it appears that GS3 had been interpreting PokeMinn's bets strangely. GS3 had been built with only 4 betting options: fold, call, bet the size of the pot and go all-in. Whenever PokeMinn made a bet larger than the size of the pot, GS3 had to decide whether to interpret that bet as a pot bet or an all-in bet. Due to PokeMinn's design, it regularly made bets of over 100 chips, followed by going all-in. GS3 interpreted the first bet as a pot bet (despite the real size of the pot being 12 chips or smaller) and it would call, but it would later fold to the all-in bet. This allowed PokeMinn to regularly win hundreds of chips from GS3.

As the tournament coordinator said, "If [PokeMinn] had been making the bets GS3 thought it was, GS3 would have crushed it." As PokeMinn was generally a weak player, the fact that it

could exploit a much *better* player so well meant that there were issues in this game people were not considering. It may be true that the underlying strategy for GS3 was very strong, but since it interpreted the real game state poorly it could be exploited for a large number of chips. Thus, I set out to investigate this situation and determine what was happening.

1.3 Contribution

This dissertation focuses mainly on the implementation of poker agents created at the University of Alberta. Most of the concepts discussed are generalizable to other games. The basic concept for creating poker strategies is shown in Figure 1.1. We wish to compute a full game strategy, but due to the size of the full game this cannot be done directly. Instead, we walk through three steps. We first create an abstraction of the full game. Second, we compute a solution to the abstract game. Third, we translate between the full game and the abstract game to use the abstract solution to play the real game.

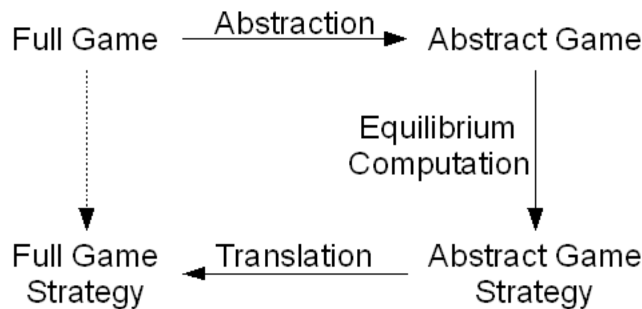

Figure 1.1: Full game strategy development process

Abstraction: A smaller version of poker is created by abstracting the full game. One method for creating the abstraction is to bucket the possible card combinations according to how good they are (for instance, according to some *hand strength* metric). Another method is to restrict what actions are legal, thus reducing the size of the game tree.

Equilibrium Computation: The abstract game is *solved* by computing a Nash equilibrium. A variety of algorithms for doing this will be discussed in Section 2.2. This computation results in a strategy for the abstract game.

Translation: States in the real game must be translated into states in the abstract game in order to use the strategy for the abstract game in the real game. This can be a simple mapping of real states to abstract states or a more complicated process. This enables one to use the abstract strategy to play in the full game.

This work focuses on translation, the final step. As was seen in Table 1.1, a strong player with poor translation can result in a very exploitable player. The concept of translation will be formally defined and analyzed, including an in-depth examination of the no-limit agents produced by the

University of Alberta. We also design a number of exploitative techniques that take advantage of agents that employ poor translation methods. In designing these techniques, we are able to find new translation methods that defend against exploitative agents. These methods come in the form of soft translation, a probabilistic translation method that results in a smooth view of the action space, and strategy switching, a translation method that takes advantage of strategies with different underlying abstractions. Both of these methods are shown to produce more robust players than the current translation methods being used.

In addition to the work on translation, this dissertation explores the no-limit aspect of poker strategies. Many of the techniques used for limit poker, including DIVAT [2] and importance sampling [4], are extended to no-limit and examined. The abstraction methods used by limit poker are also investigated, and an extensive exploration of possible abstraction choices for no-limit is performed. This investigation provides us with a better understanding of the important aspects of no-limit poker and will help us create an agent capable of competing with champion level human players.

Chapter 2 discusses the underlying concepts in game theory revolving around abstraction and equilibria. It also shows some brief results from extending evaluation techniques used in Limit Texas Hold'em to the no-limit game. Chapter 3 analyzes how the no-limit Polaris agent works, laying the ground work for important aspects of translation. Chapter 4 formalizes the concept of translation and shows how this concept applies to poker. Additionally, it provides a new method of translation that creates more robust players than before. Finally, Chapter 5 investigates the concept of switching, within a hand, between strategies that use several different abstractions to create a player with a better understanding of the real game.

Chapter 2

Background

2.1 Extensive Form Games

Although this work deals primarily with the game of poker, most of the algorithms and techniques described can be applied to many highly complex, competitive scenarios with imperfect information. Many such problems, including poker, can be represented using an extensive form game tree formulation. One of the reasons why this formulation is so useful is because of the power it provides in creating strategies for games, as well as the flexibility it provides in describing many different games. Given a zero-sum two player extensive form game, we know that there exists an equilibrium strategy. An equilibrium strategy is a strategy in which no player can benefit by deviating from the strategy and results in a strategy that bounds the worst case scenario. We can also evaluate strategies in terms of how far from equilibrium they are. This provides us with an excellent metric for evaluating strategies, in that we can calculate how much a strategy can be beaten by. All of these concepts will be formalized below.

2.1.1 Definition

An extensive game involves combinations of actions taken by players and chance. For example, in poker, the actions would be the player actions (fold, call or raise) together with the chance actions (cards dealt). Each list of actions is called a history and hidden information can be modeled by partitioning the histories into sets, called information sets, whose elements cannot be distinguished from one another by the acting player. For example, in poker, two histories that differ only by the opponent's cards would be indistinguishable and would be in the same information set. Each history also has a player or chance assigned to it, designating whose turn it is to act.

Formally, we can define an extensive game as follows.

Definition 1 (Extensive Game) *[23, p. 200] A finite extensive game with imperfect information is denoted Γ and has the following components:*

- *A finite set N of* ***players****.*

- *A finite set H of sequences, the possible* ***histories*** *of actions, such that the empty sequence is in H and every prefix of a sequence in H is also in H. $Z \subseteq H$ are the* ***terminal histories****. No sequence in Z is a strict prefix of any sequence in H. $A(h) = \{a : (h, a) \in H\}$ are the actions available after a non-terminal history $h \in H \setminus Z$.*

- *A* ***player function*** *P that assigns to each non-terminal history a member of $N \cup \{c\}$, where c represents chance. $P(h)$ is the player who takes an action after the history h. If $P(h) = c$, then chance determines the action taken after history h. Let H_i be the set of histories where player i chooses the next action.*

- *A function f_c that associates with every history h for which $P(h) = c$ a probability measure $f_c(\cdot|h)$ on $A(h)$. $f_c(a|h)$ is the probability that a occurs given h, where each such probability measure is independent of every other such measure.*

- *For each player $i \in N$, a partition $\mathbf{I}_i$ of H_i with the property that $A(h) = A(h')$ whenever h and h' are in the same member of the partition. $\mathbf{I}_i$ is the* ***information partition*** *of player i; a set $I_i \in \mathbf{I}_i$ is an* ***information set*** *of player i.*

- *For each player $i \in N$, a* ***utility function*** *u_i that assigns each terminal history a real value. $u_i(z)$ is rewarded to player i for reaching terminal history z. If $N = \{1, 2\}$ and for all z, $u_1(z) = -u_2(z)$, an extensive form game is said to be* ***zero-sum****.*

In this work we focus exclusively on two player zero-sum games. Many games also work under the assumption of *perfect recall*. Perfect recall refers to the fact that a player remembers every action that was taken, by both players and chance, during the game.

We can define a strategy σ for an extensive game as a probability distribution over legal actions for every possible history.

Definition 2 (Strategy) *[32] A* ***strategy of player i*** *σ_i is a function over $A(I_i)$ for each $I_i \in \mathbf{I}_i$ and Σ_i is the set of all strategies for player i. A* ***strategy profile*** *σ consists of a strategy for each player, $\sigma_1, \sigma_2, ...$, with σ_{-i} referring to all the strategies in σ except σ_i.*

Let $\pi^\sigma(h)$ be the probability of history h occurring if players choose actions according to σ. We can decompose $\pi^\sigma = \Pi_{i \in N \cup \{c\}} \pi_i^\sigma(h)$ into each player's contribution to this probability. Hence, $\pi_i^\sigma(h)$ is the probability that if player i plays according to σ then for all histories h' that are a proper prefix of h with $P(h') = i$, player i takes the corresponding action in h. Let $\pi_{-i}^\sigma(h)$ be the product of all players' contribution (including chance) except player i. For $I \subseteq H$, define $\pi^\sigma(I) = \sum_{h \in I} \pi^\sigma(h)$, as the probability of reaching a particular information set given σ, with $\pi_i^\sigma(I)$ and $\pi_{-i}^\sigma(I)$ defined similarly.

If we have a strategy for a game, then we can create an agent that plays the game according to that strategy. A **strategy player** is a player that, when it is player i and $h \in I_i$, takes actions by sampling

from $\sigma_i(I_i)$. If all players play according to some strategy σ, then we can define the expected utility for player i to be $u_i(\sigma)$. This expected utility can be directly calculated because having σ allows us to calculate the probability of reaching each terminal node, and thus the expected value is simply the weighted sum of the utility of the terminal nodes. Using the above notation, we find that $u_i(\sigma) = \sum_{h \in Z} u_i(h)\pi^{\sigma}(h)$. We will often use the notation $u_i(\sigma_i, \sigma_j)$ to refer to $u_i(\{\sigma_i, \sigma_j\})$.

2.1.2 Poker Example

Poker can be described as a zero-sum extensive form game. Due to the size of most poker games, it is difficult to explicitly describe them. However, we can do so using heads-up Kuhn poker [15]. Kuhn poker is a simple game in which there are only three cards, the jack, queen and king. Each player antes one chip and receives one private card. The first player can then check (k) or bet (b). If the first player checked, the second player can also check or bet, otherwise the second player can call (k) or fold (f). If the second player bets, the first player can also call or fold. If neither player has folded, the player with the higher card wins the pot. We can see how this is formulated as an extensive game.

Definition 3 (Kuhn Poker) *A formalization of the Kuhn Poker extensive form game is as follows:*

- *N = $\{1,2\}$*
- *All histories $h \in H$ are of the form $\{\emptyset\}$, $\{c\}$ or $\{ca_1...a_n\}$ where c are the cards dealt to the players and can be any of the following: $c_{KQ}, c_{KJ}, c_{QK}, c_{QJ}, c_{JQ}, c_{JK}$. $a_1...a_n$ is the betting sequence and can be $k, b, kk, kb, kbk, kbf, bk, bf$. $Z \subseteq H$ are the histories kk, kbk, kbf, bk, bf.*
- *$P(\emptyset) = chance$, $P(c) = 1$, $P(ca_1) = 2$, $P(ca_1a_2) = 1$*
- *f_c is evenly distributed over the options and is thus $1/6$ for all options*
- *Each player cannot see the private card of the other player. Thus, $\mathbf{I}_1$ has any history with the prefix c_{KQ} and the history that differs by changing the first action to c_{KJ} in the same block of the partition. The same holds true for c_{QK} and c_{QJ} as well as c_{JQ} and c_{JK}. Player 2 exhibits a similar property with the cards reversed. All other differences in the histories result in being in different information sets.*
- *We have $u_1(cbf) = 1$, $u_1(ckbf) = -1$, $u_1(ckk) = 1$ when player 1 is dealt the higher card and -1 otherwise. For $z \in \{ckbk, cbk\}$, $u_1(z) = 2$ when player 1 is dealt the higher card and -2 otherwise.*

One can imagine how this formalization could extend to larger games of poker. More cards can be dealt, cards can be dealt multiple times, and more actions can be taken by the players.

2.1.3 Best Response and Nash Equilibria

If we have some strategy σ, then we can calculate the utility for player i, $u_i(\sigma)$, given that all players play according to σ. If we fix σ_i and vary the other player strategies to create σ', the new utility $u_i(\sigma')$ may be different. Ideally, we would like some guarantee on what utility player i will obtain when using strategy σ_i, which brings us to the concept of a Nash Equilibrium [18].

Definition 4 (Nash Equilibrium) *A **Nash equilibrium** is a strategy in which no player can benefit by deviating from the strategy. Thus, σ is a Nash equilibrium if and only if for all players i:*

$$u_i(\sigma_i', \sigma_{-i}) \leq u_i(\sigma_i, \sigma_{-i}) \quad \forall \, \sigma_i' \tag{2.1}$$

*An **ϵ-Nash equilibrium** is a strategy σ where each player is at most ϵ away from an equilibrium, or for all players i:*

$$u_i(\sigma_i', \sigma_{-i}) \leq u_i(\sigma_i, \sigma_{-i}) + \epsilon \quad \forall \, \sigma_i' \tag{2.2}$$

Nash equilibria are strategies that are minimally exploitable. Exploitability is defined as how much a player could take advantage of another player's strategy. For instance, if we consider player i's strategy to be a set of variables, we can calculate the strategy that maximizes player i's utility, given that all other players play according to σ. Thus, if we let σ_i vary but keep all other player strategies within σ constant, we can find the best response for player i to σ_{-i}.

Definition 5 (Best Response) *Given some strategy σ_{-i} the **best response** to σ_{-i} for player i is the strategy σ_i that maximizes player i's utility, having value:*

$$br_i(\sigma_{-i}) = \max_{\sigma_i \in \Sigma_i} u_i(\sigma_i, \sigma_{-i}) \tag{2.3}$$

We observe that if σ^* is an equilibrium strategy, then $br_i(\sigma^*_{-i}) = u_i(\sigma^*)$. If σ is not an equilibrium strategy, then we say that $br_i(\sigma_{-i}) - u_i(\sigma^*)$ is the exploitability of σ_{-i}. Thus, we can say that a strategy σ is an ϵ-Nash equilibrium if $br_i(\sigma_{-i}) - u_i(\sigma^*) < \epsilon \, \forall \, i$. In two player games, the best response tells us how far a player's strategy is from equilibrium. In zero-sum games, we have that $\sum_i u_i(\sigma^*) = 0$, and thus $\sum_i br_i(\sigma_{-i})$ is the exploitability of the strategy profile σ.

2.2 Equilibrium Algorithms

Nash equilibria guarantee an upper bound on the exploitability of an agent. In two player zero-sum games, the expected utility of an equilibrium strategy profile is 0. John Nash proved that every two player zero-sum game has an equilibrium solution [18]. The question is then how to find an equilibrium in a given game, and a variety of techniques have been developed over the years to do so. *Linear programming* is a concept that was developed in 1963 [6] and still proves to be useful today. However, new methods have arisen that allow one to solve larger games. Two families of methods include *gradient-based algorithms* [7] and *counterfactual regret minimization* [32].

2.2.1 Linear Programming

Linear programming is a technique that has been around since 1963 [6]. The general concept is that we wish to maximize a real-valued affine function f:

$$f(x_1, x_2, ..., x_n) = a_1x_1 + a_2x_2 + ... + a_nx_n + b \tag{2.4}$$

or equivalently a^Tx, with some constraints on the vector $x = \{x_1, ..., x_n\}$, which can be specified as $Ax \leq c$ and $x \geq 0$, where A is a matrix of constraint coefficients and c is a vector. Once a problem has been formalized as a linear programming problem, it can then be solved in polynomial time using a number of methods including the *simplex method* [19] and the *interior point method* [17].

Poker can be formulated as a linear programming problem using the sequence form representation of a strategy [29]. Given a two player game with strategy $\sigma = \{\sigma_1, \sigma_2\}$, we know that the utility for a player can be calculated as $u_i(\sigma) = \sum_{h \in Z} u_i(h)\pi^\sigma(h)$. Since we know σ, $\pi^\sigma(h)$ is simply the product of chance node and strategy probabilities. We can then write $\pi^\sigma(h) = \pi_1^\sigma(h)\pi_2^\sigma(h)\pi_c^\sigma(h) \,\forall\, h$, and $u_i(\sigma) = \sum_{h \in Z} \pi_1^\sigma(h)\pi_2^\sigma(h)\pi_c^\sigma(h)u_i(h)$. We note that all $\pi_c^\sigma(h)$ and $u_i(h)$ are constants, and thus we only need to solve for $\pi_1^\sigma(h)$ and $\pi_2^\sigma(h)$. Let x and y represent vectors of all possible sequences for players 1 and 2, respectively, including the null sequence. Define A to be the expected payoff matrix for player 1, where $A_{ij} = \pi_c^\sigma(h)u_1(h)$ if following sequences i and j by players 1 and 2 results in a valid terminal history $h \in Z$, and $A_{ij} = 0$ otherwise. The game then takes the form of x^TAy, where player 1 is attempting to maximize this value and player 2 is attempting to minimize this value. Finally, we have the constraints on x and y in the form of $Ex = e$, $Fy = f$, $x \geq 0$ and $y \geq 0$. The constraints require that the sum of the probabilities leading into an information set equals the sum of the probabilities leaving the information set. For instance, if we have a sequence s with an associated weight w and the set of sequences S having s as their immediate prefix, then the sum of the weights of the sequences in S must be w. This applies to the constraints for both x and y. The problem is thus defined in a way that is solvable using linear programming techniques. When described in this fashion, the memory required to run the algorithm is linear in the number of states in the game. This contrasts to newer methods, which only require memory linear in the number of information sets in the game. For our experiments, we use CPLEX [12], a linear programming solver, to solve smaller games of poker including Leduc Hold'em.

2.2.2 Gradient-Based Algorithms for Finding Nash Equilibria

Gradient-based algorithms for finding Nash equilibria is an approach described by Gilpin and colleagues [7]. Gradient-based algorithms *descend* to the equilibrium point in an extensive game from an arbitrary starting point. To start, they consider the saddle-point problem that follows from the

sequence form representation of a two player zero-sum sequential game:

$$\max_x \min_y x^T Ay = \min_y \max_x x^T Ay \tag{2.5}$$

The problem with using gradient-based algorithms here is that $\min_y x^T Ay$ and $\max_x x^T Ay$ are not smooth functions. Thus, they must first be smoothed before any work can be done.

Gilpin and colleagues use the excessive gap smoothing technique [20] to smooth the aforementioned functions. This technique provides them with prox functions d_1 and d_2 that allow them to create smooth versions of the previous functions: $min_y x^T Ay + u_2 d_2(y)$ and $max_x x^T Ay - u_1 d_1(x)$ for some $u_1, u_2 > 0$. Their algorithm is then designed to reduce u_1 and u_2 over time as x and y converge to approximate solutions. These solutions are then ϵ-Nash equilibria in the original game.

2.2.3 Counterfactual Regret Minimization

Counterfactual regret minimization is a technique first proposed by Zinkevich *et al.*[32] and Johanson [13]. This technique uses regret minimization to eventually converge to an equilibrium. The concept of **regret** is that after performing some actions, we can calculate how much we would rather have performed some other actions. Specifically, if we play one game according to strategy σ and obtain some utility u, our regret is $u^* - u$, where u^* is the maximum utility we could have obtained by playing some other strategy (though this strategy must be the same when repeatedly playing the game). If a game is played repeatedly, we can define the average regret to be the average of the regret for every play. If the average regret of all the players in a game is less than ϵ then the strategy is an ϵ-Nash equilibrium [32, page 3].

The important concept of CFR is to calculate a regret at every information set instead of over the whole strategy space. Given strategies σ^t for each time step t, information set I and utility of playing σ^t given we reached set I $u_i(\sigma^t, I)$, then the **immediate counterfactual regret** is [32, page 4]:

$$R^T_{i,imm} = \frac{1}{T} \operatorname*{argmax}_{a \in A(I)} \sum_{t=1}^{T} \pi^{\sigma^t}_{-i}(u_i(\sigma^t|_{I\to a}, I) - u_i(\sigma^t, I)) \tag{2.6}$$

where $\sigma^t|_{I\to a}$ represents σ^t where player i performs action a at information set I. They prove that minimizing the counterfactual regret at every information set also minimizes the overall regret, and thus leads to an equilibrium solution when played in self-play. CFR is the method we use to create our agents in the Texas Hold'em game.

2.3 Abstraction

Since many games are too large to handle in their entirety, abstraction is often used to make the game a manageable size. Large games can be abstracted by merging information sets to reduce their total number. Since an information set contains histories and a history is a sequence of player and chance actions, there are several techniques for increasing the size of an information set. One technique

combines chance actions together into *buckets*. For example, in poker, multiple player hands could be combined together into a single bucket. A second technique artificially reduces the number of allowable player actions in the abstraction. For example, in no-limit poker, a raise could artificially be constrained to be the amount currently in the pot (*pot*) or the current player's full stack (*all-in*).

2.3.1 Definitions

More formally, game abstraction is defined as follows.

Definition 6 (Abstraction) *[30] An* ***abstraction for player i*** *is a pair $\alpha_i = \langle \alpha_i^{\mathbf{I}}, \alpha_i^A \rangle$, where,*

- *$\alpha_i^{\mathbf{I}}$ is a partitioning of H_i, defining a set of abstract information sets that must be coarser[1] than $\mathbf{I}_i$, and*
- *α_i^A is a function on histories where $\alpha_i^A(h) \subseteq A(h)$ and $\alpha_i^A(h) = \alpha_i^A(h')$ for all histories h and h' in the same abstract information set. We will call this the abstract action set.*

The ***null abstraction*** *for player i, is $\phi_i = \langle \mathbf{I}_i, A \rangle$. An* ***abstraction*** *α is a set of abstractions α_i, one for each player. Finally, for any abstraction α, the* ***abstract game****, Γ^α, is the extensive game obtained from Γ by replacing $\mathbf{I}_i$ with $\alpha_i^{\mathbf{I}}$ and $A(h)$ with $\alpha_i^A(h)$ when $P(h) = i$, for all i.*

Waugh *et al.*[30] did an analysis of the effect of abstracting games. In particular, they found that monotonicity in abstraction refinement does not hold. Assume we have two abstractions α_a and α_b of Γ such that α_a is a strict refinement of α_b, so that every information set in α_b is the union of some information sets in α_a. This means that α_a holds strictly more information about the real world than α_b. Waugh found that an equilibrium solution to Γ^{α_a} could be more exploitable in Γ than an equilibrium solution to Γ^{α_b}. In essence, larger abstractions do not necessarily produce better strategies.

This definition of abstraction runs into some problems when performing action abstraction. In no-limit poker, players have possibly hundreds of betting options, which makes the game incredibly large (the 500BB game has roughly 10^{71} game states [9]). Creating and storing the partitioning for this space would be infeasible, and thus we want a type of abstraction that does not require us to explicitly specify the partition. Additionally, if we wish to restrict which actions are legal, then suddenly a large number of histories no longer make sense. If bets of size 2 are disallowed, then a normal abstract game would still have histories in which these bets occur. However, there is no need for the game definition to contain histories in which a bet of size 2 occurs. If we were only to maintain the legal histories according to the abstraction definition, then the number of histories would be much smaller. Therefore, we define a more general kind of abstraction, called a **loose abstraction**, that provides more flexibility.

[1] Recall that partition A is coarser than partition B, if and only if every set in B is a subset of some set in A, or equivalently x and y are in the same set in A if x and y are in the same set in B.

Definition 7 (Loose Abstraction) *An extensive game Γ' is a **loose abstraction** of Γ if $H' \subseteq H$ and $\exists$ an abstraction α such that $\forall i$*

- *$\exists$ a bijection between $\mathbf{I}'_i$ and $\alpha_i^{\mathbf{I}}$ and any two histories $h'_1, h'_2 \in H'$ in the same information set in $\mathbf{I}'_i$ are in the same information set in $\alpha_i^{\mathbf{I}}$*
- *$A'(h') = \alpha_i^A(h') \; \forall \; h' \in H'$*

The concept of a loose abstraction is that there exists a partitioning of the full game space, but we are not explicitly specifying it. We see that this is a more flexible type of abstraction in that for any abstraction α, Γ^α is a loose abstraction of Γ. This definition allows us to define an abstract game based solely on restricting a specific set of actions from the real game, without specifying how the other actions are handled. However, to use an abstract game strategy to play in the real game we must find a way to handle the histories in the real game that contain these restricted actions. This is generally done with translation, which is the focus of this work.

2.3.2 Card Abstraction

There are a variety of different methods that have been used to abstract the cards in poker, but all of them revolve around bucketing. If we have n available buckets, we must decide how to partition the full card space into said buckets. In general, we wish to place *similar* hands in the same bucket, for some metric of *similar*. Thus, card abstraction typically boils down to different methods of clustering hands. When doing so, there are a variety of different bucket types one can focus on.

Bucketing Options

If a player never forgets something it once knew, it is said to have **perfect recall**. When considering bucketed games, this means that the player remembers what bucket it was in on every round. Perfect recall has some benefits because of the guarantees it provides with respect to equilibrium solving. However, it restricts how many buckets one can have on each round, as the total number of buckets is the product of the number of buckets on each round. In particular, it means that the total number of bucket sequences for every round is strictly increasing, such that an abstraction with 8 buckets on every round actually has 8 buckets on the preflop, 64 bucket sequences on the flop, 512 on the turn, and 4096 on the river. While remembering earlier round buckets helps differentiate the current situation, it means we cannot have a large number of buckets on earlier rounds. If we want to increase the number of buckets on the preflop, this also increases the number of buckets everywhere else in the abstraction. A solution to this problem is to drop the perfect recall assumption.

Imperfect recall refers to having an abstraction in which the player forgets information it previously knew. For instance, if our player forgets what bucket it was in on the preflop once the flop comes, it is said to have imperfect recall. While imperfect recall allows us more flexibility in abstraction creation, using it means that we are no longer guaranteed to converge to an equilibrium. In

practice, this has not been an issue. Imperfect recall allows us to increase the number of buckets on earlier rounds without increasing the number of buckets in later rounds. For instance, in the preflop there are only 169 possible 2-card combinations one could have, taking into account suit isomorphisms. Using imperfect recall, we could create a player that has 169 buckets on the preflop that are forgotten once the flop is dealt. This could increase the number of flop buckets to 64, since the 8 buckets originally used to remember the preflop may now be used for the flop. Keeping 8 buckets on the turn and river would mean that the river still only has 4096 bucket sequences. However, we now have complete information about our cards on the preflop. The ability to increase the number of buckets on earlier rounds also gives us the ability to include different types of buckets.

Bucketing Methods

One bucketing method is potential-aware automated abstraction [8]. This method uses a bottom-up approach. In order to find the buckets for the preflop, it begins by clustering the river hands according to hand strength. It then clusters the hands on the turn according to the possible river hands they can become. For every cluster a set of histograms over possible future clusters is defined. The distance between two clusters is the L2-distance of the histograms of future clusters. After the turn buckets are assigned, the same type of clustering is repeated for the flop and finally preflop. In order to obtain the flop buckets, a similar method is employed with the restriction that for each preflop bucket, only hands that are possible children of that bucket are considered. Doing this ensures that the agent has perfect recall. This process is then repeated for the turn and river rounds to obtain the final abstraction.

Another bucketing method is *percentile bucketing* [13, page 26]. This method works by distributing all of the hands into the n available buckets according to a hand strength metric, with the top $1/n\%$ hands going into the first bucket and so on. *Hand strength* refers to the percentage of all hands that a particular hand would beat. Two hand strength metrics percentile bucketing can use are *expected hand strength* (hs) and *expected hand strength squared* (hs2) [13, pages 22-28]. For any given hand, we can roll out all of the remaining cards to find all possible future hands it could become (and the probabilities of those hands occurring). We can then compute the expectation of the final hand strength (or square of) over all these possible hands. Using perfect recall, the buckets for each round are dependent upon previous round buckets. For instance, if the flop buckets depend upon the preflop buckets, then each flop bucket actually consists of the pair [preflop bucket, raw flop bucket]. When using imperfect recall, this is not necessarily the case in that the flop buckets consist only of flop data.

Public buckets refer to buckets that specify what type of public information is available. Previously, once the flop came we would bucket our hand based upon its hand strength. Public buckets allow us to cluster the board cards, providing the player with information about the *texture* of the board. For instance, some boards may have more draw capabilities or have paired cards, whereas

others may have few opportunities for high strength hands. This can greatly affect how strong two hands that have equal strength (according to hs2) actually are. To create public buckets we first create an abstraction. A transition table is then created between rounds for each board, in that it shows the probability of transitioning from each bucket in round A to each bucket in round B given the board. These tables are then used as input to the K-Means clustering algorithm [10], which gives us the clusters of boards that can be used as public buckets.

Notation

The card abstractions we use are a combination of some of the aforementioned techniques, in particular, percentile bucketing and public buckets with and without perfect recall. There are several types of notation we use to describe the actual abstraction used. *Nhs* and *Nhs2* refer to having N hand strength or hand strength squared buckets. *hs2* is assumed to be the bucket type if none is specified, except on the river in which hs is used because it is equivalent to hs2 at that point. For instance, *8.8.8.8* refers to the perfect recall player that has 8 hs2 buckets on every round. Occasionally we will refer to abstractions as being *8s sized*, which means the abstraction is roughly the same size as the 8.8.8.8 abstraction (in terms of total game states). When using imperfect recall, *irN* specifies that there are N buckets allotted on this round, but they are forgotten in future rounds. For instance, *ir169.64.8.8* refers to the imperfect recall player that has perfect preflop buckets that are forgotten, and then the appropriate hs2 buckets to be 8s sized on the remaining rounds. It has 64 hs2 buckets on the flop because the perfect recall version would be remembering 8 preflop buckets, and combined with the 8 flop buckets makes 64 total buckets. *pubN* refers to having N public buckets, and when nesting different types of buckets we refer to them as *YxZ*, where Y and Z are types of bucket abstractions. Finally, *dir-Y-Z* specifies that we know Y for this round, but only remember Z in later rounds. Thus, *ir169.dir-pub20x36hs2x5hs-pub20x1.dir-pub3x20hs2x3hs-pub3x1.60* refers to the player that has a perfect preflop that is forgotten, a flop that nests 20 public buckets with 36 hs2 buckets and 5 hs buckets but only remembers the 20 public buckets, a turn that nests 3 new public buckets with 20 hs2 buckets and 3 hs buckets but only remembers the public buckets, and a river with 60 hs buckets. This allows us to create a wide variety of abstractions to use for experiments and competitions.

2.3.3 Action Abstraction

The action abstraction we use works by restricting the number of actions a player can take. The method used by many researchers and first published by Gilpin *et al.*[9] limits every player to 4 actions: fold (*f*), check/call (*c*), raise pot (*p*), or go all-in (*a*). Raising pot refers to making a bet of the size of the number of chips in the pot, and going all-in refers to betting all of the chips in one's stack. This is an abstraction of the full game in which the actions are restricted to f, c, p, and a, referred to as *fcpa*. If we wish to add other betting options, we need only assign them a different

letter. For instance, we refer to a bet of twice the size of the pot as a double-pot bet (*d*). The betting abstraction used is then the concatenation of the letters representing legal actions, generally in increasing order. Unlike with card abstraction, we do not cluster all the real states into abstract states. Instead, we rely on translation between the full game and the abstract game to handle bet sizes that are not legal in the abstract game. The translation process is described in more detail in the following chapters.

2.4 Evaluation

We have seen that there are a large number of ways to create poker agents, most of which come from the ways one can abstract the game. However, as we will see later, how we translate action sequences also drastically affects the play of our agents. Thus, when trying out a new abstraction type or new translation method, we wish to be able to evaluate how well the agent using it performs. This allows us to compare how an agent using some method performs in comparison to an agent using a different method.

One evaluation method is to calculate the best response to the player. This method is an excellent metric because it tells us exactly how exploitable a player is. For this reason we regularly create players in the Leduc Hold'em game that implement the techniques we are evaluating, since Leduc is small enough to calculate the best response to our players. Although this is an excellent metric for the strategies it evaluates, we have the problem that the smaller game may not properly model the situations we encounter when playing the larger game of Texas Hold'em. Unfortunately, all Texas Hold'em variants we use are too large to compute best responses in a timely manner. This means that we have to use other methods to evaluate our players.

The most obvious method for evaluation is to play our agents against other agents and record the outcome. Although winnings is a good indicator of which player is better, sometimes we may not be able to play enough hands to obtain statistical significance in a match. We can use DIVAT [2] to obtain a better estimate. DIVAT is a technique that attempts to factor out the inherent luck in the game. We can also use importance sampling [4], which allows us to evaluate hands the players never actually played. Both of these techniques provide us with the ability to more accurately evaluate poker play than using direct play on randomly dealt hands.

2.4.1 Direct Play

The normal method for evaluating whether one player is better than another is to simply have them play hands of poker against each other. However, when we do so we have to deal with the fact that there is a large amount of variance in the game. This means that we often have to play many hands in order to properly evaluate a match. One method we can use to reduce this variance is to play duplicate matches. A **duplicate match** refers to playing two matches with the same cards, except the players are in different seats in each match. The score for each hand is then defined to be the

sum of the scores for that hand in each match. This ensures that each player is put into the same situations as the other, and thus luck plays a smaller role. This works well for computer agents since we can ensure that they have no memory of the other side of the match (or the cards that would be coming).

	Bankroll	Std	Duplicate Std
Bluffbot vs Hyperborean	-0.11	18.04	11.38
Bluffbot vs Tartanian	0.61	31.46	20.24
Tartanian vs Hyperborean	-0.63	32.46	20.50
Ballarat vs Bluffbot	-2.57	45.30	30.70
Ballarat vs Hyperborean	-2.13	44.18	29.95
Ballarat vs Tartanian	-5.37	73.20	48.00

Table 2.1: Money variance analysis of the 2008 no-limit competition in sb/h

Table 2.1 shows an analysis of the 2008 no-limit competition. The first column shows the match being evaluated, with each match being between two of the four competitors: Bluffbot, Hyperborean, Tartanian and Ballarat. The third column shows the standard deviation of the money won/lost each hand, assuming each hand is independent of the other hands. The last column shows the duplicate standard deviation, which assumes that the value for each duplicate hand is the sum of the values for the two corresponding hands. Looking at the table, we observe two things of importance. First, we see that the standard deviation varies wildly from 18 sb/h to 73 sb/h for non-duplicate hands. Since players have more control over the pot size, this means that the variance of the match will be much more dependent upon the players in the match than it is in limit poker. Second, we see that the duplicate standard deviation is roughly $2/3$ of the regular standard deviation, regardless of the match. It seems that duplicate matches provide a reduction of roughly $1/3$, which is good but small when considering how much the standard deviation is dependent upon the players in the match.

When performing a match between two of our agents we generally wish to have a very accurate measure as to the difference in their ability. Thus, unless otherwise specified, all of the agent-agent matches described in this work play 10 duplicate matches of 500,000 hands each, resulting in a total of 10,000,000 hands being played. This provides us with very accurate measurements. In the 500BB game, results generally have a standard deviation of around 15-25 mb/h. In the 100BB and 200BB games, the standard deviation is near 5-10 mb/h and 10-20 mb/h, respectively.

There are certain instances in which we wish to evaluate two different techniques a player could employ, but we cannot simply play them against each other. For example, when testing a technique that deals with translation between the real action space and the abstract action space, playing agents that use different techniques against each other will not help us if they only take actions that are legal actions in the abstract game. In this situation we need to create other agents that can test the bounds of these techniques. A suite of agents can then be played against the different players we create, and we can evaluate the players based upon how well they perform relative to each other. Essentially, because the game is too large to compute a best response, we create a suite of agents that we hope

reasonably tests the exploitability of the agents.

2.4.2 DIVAT

In the past, DIVAT has been used to evaluate heads-up limit matches [2, 14]. DIVAT is an estimator that works by attempting to calculate the skill displayed in a match by removing the most obvious factors of luck. It does this by comparing a player's actions against a baseline strategy and assigning a value of skill based upon whether it performs better or worse than the baseline strategy. The primary reason why DIVAT is a good estimator is that it has been proven to be unbiased [31]. This means that in the long term, DIVAT is expected to return the same result as the metric it is estimating, in our case money.

The concept of DIVAT is actually fairly simple. First, one obtains a baseline strategy for the game being played. In poker, one such strategy is *always-call*, in which the player chooses to check/call for every action. Although this strategy is not very realistic, it is simple, fast, and easy to understand. Having obtained a baseline strategy, one needs to look at every sequence of actions not containing chance nodes and compare the expected value of the baseline strategy against the expected value of what actually occurred. In poker, this is done on a round by round basis. For instance, on the flop we would calculate two values. First, we would play out the round as if each player played using the baseline strategy. We would then calculate the expected value of the resulting state (this is done by rolling out the remaining chance nodes assuming both players play according to the baseline strategy), which we will refer to as $V(Baseline)$. Second, we play out the round according to the actual actions and again calculate the expected value of the resulting state in the same manner as before, which we will refer to as $V(Actual)$. The DIVAT score for the round is then $V(Actual) - V(Baseline)$. By repeating this for every round in the game and summing up the results, we obtain the DIVAT score for the hand.

DIVAT has been shown to be very effective at reducing the variance in limit matches. Most heads-up limit matches have a standard deviation of around 6 sb/h, and Kan *et al.*were able to achieve standard deviations as low as 1.93 sb/h [2] using DIVAT. This was using a *bet-for-value* baseline strategy that logically folded bad hands and raised with good hands. Ideally we could apply the same concept to the no-limit game and obtain similar results.

	Bankroll	Std	Duplicate Std
Bluffbot vs Hyperborean	-0.12	15.48	9.99
Bluffbot vs Tartanian	0.63	24.76	16.49
Tartanian vs Hyperborean	-0.63	28.84	18.65
Ballarat vs Bluffbot	-2.52	35.35	24.59
Ballarat vs Hyperborean	-2.11	37.65	25.77
Ballarat vs Tartanian	-5.33	63.50	43.15

Table 2.2: DIVAT variance analysis of the 2008 no-limit competition in sb/h

Table 2.2 shows the DIVAT analysis of the 2008 no-limit competition, using always-call as the

baseline strategy for DIVAT. A simple no-limit *bet-for-value* strategy was created to be used as a baseline strategy, but it took much longer to run and only produced marginally better results than using always-call. Looking at the table, we see that the values are fairly similar to the ones in Table 2.1. Overall, DIVAT appears to reduce the standard deviation by 10-20%. This is somewhat disappointing, as it is nowhere near the same reduction that was achieved in the limit game. We believe that since the players have more control over the pot size and it is less common to reach a showdown than in limit poker, that a more advanced baseline strategy would be needed to obtain better results.

2.4.3 Importance Sampling

Importance sampling is another method for reducing variance in player evaluation when one of the player's strategy is known explicitly. The key aspect to importance sampling is that it allows us to use situations we did not observe, but are similar to what we did observe, to evaluate players [4]. This means that we obtain many more samples than originally available. Importance sampling also allows us to evaluate the strategy of players not in the observed match. This allows us to estimate how well another player would perform against a particular opponent without actually playing that player in the match.

The concept of importance sampling is that we can compute the expected value of some value function V given a set of observations O, a strategy σ used to produce O, and another strategy σ'. This is done by calculating the probabilities of each $h \in O$ occurring given that either σ or σ' were used to play. The ratio of these probabilities is used to weight the observed value $V(h)$, and the sum of the weighted values gives us the expected value. *On-policy* importance sampling refers to when we are evaluating the observed strategy, or $\sigma = \sigma'$. *Off-policy* refers to when we are evaluating a different strategy than the observed one, or $\sigma \neq \sigma'$.

We refer to using *basic* importance sampling when it is only run on O. However, since we have σ and σ' we can evaluate other situations as well. If for some observation $h \in O$ the player could have taken an action that would result in no more opponent actions, then we can evaluate the history in which that action occurred. This is called *game ending actions* (GEA) and occurs in poker whenever our player could have folded, called a bet on the river, or (in no-limit) called an all-in bet. Additionally, we can evaluate situations that our opponent cannot distinguish from $h \in O$. This is called *other cards* (OC) and in poker consists of switching the player's private cards with different cards, since the opponent cannot see them and thus cannot act upon them. Finally, the value function V is just the money exchanged in the hand. However, since it can be any evaluation function, we can also use DIVAT as the value function, allowing us to combine the two techniques.

Table 2.3 shows how well importance sampling reduces variance in several full-information situations. The second column is an on-policy evaluation of an 8s agent in an 8s-8s match (they use the same strategy, the number simply differentiates the players). The third column is an on-policy

IS type	fcpa.8.8.8.8_1 vs fcpa.8.8.8.8_2	Bluffbot vs Hyperborean	
	fcpa.8.8.8.8_1 (On)	Hyperborean (On)	fcpa.8.8.8.8 (Off)
Money-Basic	18.18	18.04	168.74
Money-GEA	17.98	18.11	101.49
Money-OC	14.65	14.21	56.21
Money-GEA.OC	12.61	12.86	55.38
DIVAT-Basic	15.33	15.48	99.61
DIVAT-GEA	15.03	15.76	80.69
DIVAT-OC	12.39	12.12	55.07
DIVAT-GEA.OC	10.03	11.12	54.40

Table 2.3: Importance sampling effect on standard deviation in sb/h

evaluation of Hyperborean in the Bluffbot-Hyperborean match from the 2008 competition. Finally, the last column is an off-policy evaluation of an 8s agent for the same match. It is important to note that basic importance sampling using money as the value function returns the same results as evaluating the match as normal.

Looking at the table, we see a reduction in the standard deviation as more of the options are turned on. The values for the on-policy evaluations in both of the matches are fairly similar. The values for the off-policy evaluation, however, are much higher. In the end, we are able to reduce the standard deviation of these evaluations by 45%, 38%, and 68%, respectively. This will be useful when evaluating matches in the future.

Chapter 3

Analysis of the No-limit Polaris Agent

3.1 Introduction

This chapter provides an analysis of the inner-workings of the no-limit Polaris agents. The term *Polaris* refers to the suite of agents produced by the University of Alberta Computer Poker Research Group and can encompass a wide variety of variants and strategies. Despite many similarities between limit and no-limit, creating an agent to play in the no-limit game can be quite different from the limit game. The biggest difference is the addition of action (betting) abstraction, which brings with it two new major complications. The first complication is that we now have a trade-off between making a larger betting abstraction and making a larger card abstraction for a fixed game size. The other major complication is that it is no longer a trivial task to use an abstract strategy to play in the full game. We now have to translate real action sequences into action sequences recognized by the abstract game and vice versa, which can be a dangerous process.

Section 3.2 describes the different types of abstraction we must consider. In Section 3.3 we examine exactly how Polaris performs translation, how translation has been improved and how these changes lead to the formalization of state translation. In Section 3.4, we discuss the two main translation concepts that form the basis for translation in Polaris.

3.2 Abstraction Allocation

When creating an agent for a large game we must first decide on which abstraction of the real game to use. We generally have a fixed size for the abstraction, but can vary some parameters to create it. Unlike limit games, we must consider abstractions for the action space in addition to card abstraction. With fixed computational power, we must take into account the fact that a larger betting abstraction means we must use a smaller card abstraction. Finally, when considering how to create the card abstraction, we should take into account the differences that come with the no-limit game. We may wish to rearrange the bucket assignments because of how the action space is formed.

3.2.1 Betting Abstraction

As mentioned in Section 2.3.3, we use a betting abstraction based on pot fractions. We allow our agent to make a bet of X% of the pot, where X can vary. In practice, we usually consider the options to fold, call, bet 100% of the pot, or go all-in (the *fcpa* abstraction). Additionally, we sometimes consider half the pot (*h*), three-quarter pot (*q*), 1.5 times pot (*w*), double the pot (*d*), ten times the pot (*t*), or eleven times the pot (*e*). When we add the half-pot (and three-quarter pot) option we restrict the number of times it can be used. Specifically, an agent can only perform the half-pot action once per round and not at all on the preflop. These restrictions are necessary to reduce the size of the betting tree to a tractable size. For example, the unrestricted *fchpa* tree has roughly 151 times as many nodes as the *fcpa* tree in the 500BB game. Restricting the betting in this way reduces the increase to a factor of 26. The most common betting abstractions we use in Texas Hold'em are *fcpa* and *fcpta*. When performing experiments in Leduc Hold'em, we will often use *fchpda* and all subsets produced by removing any combination of h, p, or d. This allows us to show results for a variety of betting abstractions.

Adding additional betting options generally increases the size of the betting space exponentially. In practice, the actual increase in size is dependent upon the size of the bet, the other betting options, and the stack sizes in the game. For instance, adding the restricted half-pot option to the $fcpta$ betting abstraction affects different stack sizes much differently. In the 500BB game, it increases the size by a factor of 19.0. In the 200BB and 100BB games, the increase is only a factor of 9.9 and 7.2, respectively. The question then becomes whether such an increase is worth it or not, keeping in mind that instead of adding this betting option we could increase the granularity of the card abstraction by the same amount.

To test the trade off between card and betting abstractions, we performed an experiment comparing several abstractions in the 100BB game. Two agents were created in the 8s abstraction size, one of which had the half-pot option and one did not. Another agent was created in the 12s abstraction size without the extra option. The results are shown in Table 3.1. It is important to note that the 12s abstraction is roughly 4.7 times larger than 8s, and again the $fchpta$ abstraction is 7.2 times larger than $fcpta$, making the $fchpta$ 8s abstraction 1.5 times larger than the 12s abstraction.

	(1)	(2)	(3)
(1) fchpta.ir169.64.8.8	0	20	66
(2) fcpta.ir169.72hs2x2hs.12.12	-20	0	43
(3) fcpta.ir169.64.8.8	-66	-43	0

Table 3.1: Card/betting abstraction trade-off performance of several 100BB agents in mb/h

The results suggest that having the additional betting option has a significant effect. In fact, the results appear to be proportional to the size of the abstractions. The $fchpta$ abstraction beats its $fcpta$ counterpart by 66 mb/h, roughly 1.5 times the 43 mb/h the 12s abstraction manages. However, the $fchpta$ abstraction beats 12s by 20 mb/h, which is slightly larger than we might

expect considering it is only 1.5 times larger. This suggests that the extra option is at least as good as increasing the card abstraction by the same amount. Unfortunately, in larger games (200BB or 500BB), the increase in size from extra actions may not be a feasible option.

	(1)	(2)	(3)
(1) fchpea.ir169.dir-pub20x36hs2x5hs-20x1.dir-pub3x20hs2x3hs-3x1.60	0	36	36
(2) fcpea.ir169.dir-pub20x75hs2x36hs-20x1.dir-pub3x50hs2x18hs-3x1.900	-36	0	4
(3) fchqpwea.ir169.ir72hs2x3hs.ir72hs3x3hs.216	-36	-4	0

Table 3.2: Card/betting abstraction trade-off performance of several 200BB agents in mb/h

Table 3.2 shows the results of another experiment done concerning the card/betting abstraction trade-off. Unlike the last experiment, this experiment was performed in the 200BB game and all three of these agents were specifically designed to be the same size. Thus, an agent with a betting abstraction twice as large has a card abstraction of half the size. This allows us to more directly analyze the trade-offs of changing the abstraction sizes. Strategy (2) has the smallest betting abstraction, $fcpea$, allowing only the bets of pot, eleven pot and all-in. Strategy (1) allows an additional half pot bet, making its betting abstraction roughly 15.7 times larger. Strategy (3) also allows a three-quarter pot bet and 1.5 pot bet, making its betting abstraction roughly 248.0 times larger than that of strategy (2) and roughly 15.8 times larger than that of strategy (1).

Looking at the table, we see that strategy (1) appears to perform the best, beating both of the other strategies by 36 mb/h. This is somewhat surprising, considering the high cost (a factor of 15.7) of adding in the half pot bet. Perhaps more surprising, though, is the fact that strategy (3) played as well as strategy (2) despite having a card abstraction 248 times smaller. This shows that these extra betting options really make a huge difference.

3.2.2 Card Abstraction

The addition of imperfect recall, as mentioned in Section 2.3.2, has greatly increased our options when creating card abstractions. In particular, it has given us the ability to have a perfect preflop, which refers to having a bucket for every possible preflop hand (after suit isomorphisms), and public buckets. However, the abstractions used have been focused on limit play, where most of the money comes from the turn and river rounds. In no-limit, every round has the possibility of having all of the money enter the pot. This means that it may be more important to have more buckets on every round, as well as having better knowledge of how good one's hand is at any instant. Table 3.3 shows the results of an experiment in which we created agents using different levels of recall.

Six agents were used in this experiment, with three of them being 8s-sized, two 12s-sized, and one that is 25% larger than 8s. In particular, we see that the more imperfect recall we use, the better our agents seem to perform. This is best seen by the fact that there seems to be a strict dominance of (3) over (4) over (6). We also observe that, despite being significantly smaller abstractions, the imperfect recall 8s-sized strategies ((3) and (4)) all outperform (5). As (3) performs the best out of

	(1)	(2)	(3)	(4)	(5)	(6)
(1) ir169.dir-pub20x205-20x1.pub3x68-3x1.68	0	6	20	56	108	201
(2) ir169.72hs2x2hs.12.12	-6	0	28	67	86	193
(3) ir169.ir64.ir512.4096	-20	-28	0	23	67	163
(4) ir169.64.8.8	-56	-67	-23	0	34	126
(5) 12.12.12.12	-108	-86	-67	-34	0	217
(6) 8.8.8.8	-201	-193	-163	-126	-217	0

Table 3.3: Imperfect recall performance of several 500BB fcpta agents in mb/h

this group, it seems to suggest that it is far more important to have more immediate buckets than to remember previous buckets. This concept is reflected in our current no-limit agents, as they forget all strength buckets at every round.[1] Additionally, we see that the best overall strategy appears to be (1). (1) is the same as (3) except that we increased the number of buckets on the flop and turn to match the river, giving us enough room to add public bucket information. Although this significantly increases the number of buckets on the flop and turn, it only increases the size of the abstraction by roughly 25%.[2] This change allows the strategy to keep pace with (2), a strategy roughly 4 times larger that also uses imperfect recall on the preflop. These changes have allowed us to make smaller abstractions that have better performance. Having more buckets on earlier rounds means that some of those buckets can be used for orthogonal metrics, for instance public buckets, or instead we could increase the size of the betting abstraction while still having a large number of buckets.

3.3 Polaris Translation

The concept behind the translation used by Polaris agents is fairly straightforward. Since most agents use the $fcpa$ abstraction, they can only understand four actions. Thus, any action that occurs in the real game must be interpreted as one of those four actions. Additionally, since two of the actions are not bets (fold, call), those can only be mapped to themselves. All we really need to do is to map any bet to either pot or all-in. More generally, we need to map any real bet to one of the legal abstract bets. If b is the real bet and c, d are the two closest abstract bets such that $c < b < d$, then we map b to c or d according to whether c/b or b/d is largest. For instance, if we observe a bet of size 20 and the closest legal bets are 12 and 88, then we will interpret this as a bet of 12 since $\frac{12}{20} = 0.6 > \frac{20}{80} = 0.25$. By doing this for every action in the real game, we obtain a state in the abstract game and can then use the strategy for the abstract game.

It is important to note that when this translation is done, it results in a state in the abstract game and a lot of information about the real state is lost. According to the previous example, after translating the bet of 20 into a bet of 12, the agent now believes that the bet of 12 occurred. This means that it believes that its opponent placed 8 fewer chips into the pot than what actually happened.

[1]They do however remember public bucket information, since that information does not change when we advance to further rounds.

[2]The number of betting sequences increases on each round, so increasing the number of buckets on earlier rounds has a much smaller effect than increasing the number of buckets on later rounds.

The agent acts according to the current abstract state, and thus does not understand that there are 8 more chips in the pot. This can lead to situations in which the agent has a very poor understanding of the real situation, because it believes the pot size to be drastically different than what it actually is.

After querying the strategy for an action in the abstract game, we must decide what action to perform in the real game. Polaris attempts to *fix* the pot size in the real game to match the pot size in the abstract game. For instance, if the pot size is 10 in the real game but 12 in the abstract state after translation, then the agent will add one chip to whatever action it takes (the opponent calling will add the second chip that fixes the size of the pot). Specifically, the agent looks at how many chips it is supposed to have in the pot after taking its action in the abstract state and attempts to match this number in the real state. Thus, if it believes it should have X more/less chips in the pot than it currently does, it will modify its next action to include X more/less chips (if doing so does not make it an illegal action). This tries to ensure that the real pot size will never be that different from the abstract pot size.

3.3.1 Exploitative Techniques

Unfortunately, this type of translation leads to some exploitable behavior. Since all bets must be translated to either pot or all-in, this means that there are large ranges of bets that will all be interpreted as the *pot* action. Specifically, any bet smaller than $\sqrt{p*a}$ is interpreted as a pot bet, where p is the size of the pot and a is the size of an all-in bet. This is because $\frac{\sqrt{p*a}}{a} = \frac{\sqrt{p}}{\sqrt{a}} = \frac{p}{\sqrt{p*a}}$. This means that, when making a pot bet, one could choose to bet anything up to this amount (referred to as *up-betting*) or similarly as low as the minimum bet (referred to as *down-betting*). This fact can be exploited in several ways.

The most obvious method to exploit this many to one mapping is for an agent to alter its bets based upon its hand strength. Since our agent's bucketing method already uses a hand strength metric, it can simply query the abstraction for the current bucket. Once the agent knows the bucket, it can deem each hand *good* or *bad* based on whether the current bucket is a high bucket or a low bucket. If the agent assumes the opponent folds half its hands to any bet the agent makes, this means that for the agent's hand to be *good* it needs to be in the top half of the hands its opponent would call with. Thus, if an opponent folds the bottom 50% of their hands, then an average hand assuming a call would have a strength of 75%. So if the hand is in the top 25% of buckets it is a *good* hand and in the bottom 75% it is a *bad* hand. We can now define an agent that uses this information to its advantage. This is the +- player, which up-bets with good hands and down-bets with bad hands. Doing so allows it to make the pot larger when it is more likely to win and to keep the pot smaller when it is more likely to lose.

There are two other simple agents that can take advantage of the many to one mapping without considering hand strength. The first is the *AllDown* player, which simply down-bets every bet. The

second is the *naïvePA* player. This player does not look at a strategy to decide what to do, but simply performs the same actions every hand. All it does is up-bet a pot bet, followed by down-betting an all-in bet. This is effective because it floods the pot with chips, which the opponent does not see. It then goes all-in, forcing the opponent to make a tough decision in which it incorrectly folds too often since it believes the pot is too small to call (when in reality the pot is over 10 times the size it thinks it is). It is surprising how effective this method works, particularly as the naïvePA technique does not require the agent to even look at its cards.

Finally, there are two more metrics for evaluating how well a translation method works. The first is heads-up play against the BluffBot 2.0 agent [25] that won the 2007 no-limit competition. The second is to look at the best response value for a strategy in the Leduc Hold'em game.

3.3.2 Translation Fixes

Without modifications we can get into some strange situations. One such situation is if an agent is playing against another version of itself and the pot is too small. Instead of calling, the agent will make a small bet in order to increase the pot size to the correct amount. However, the other agent will interpret this action as a bet and not a call, thus resulting in an extra bet in the abstract state and meaning that the real pot is still too small. In attempting to call, this agent will again make a small bet to fix the pot. This process repeats until an agent folds or both agents are all-in, as neither agent will ever take the call action. This is a behavior we do not want our agents to have.

The biggest problem an agent has is misinterpreting its previous actions. In the situation just described, the agent is making a call but performs a different action in the real game. When translating the real state later, we want the agent's action to be interpreted as whatever the abstract strategy told it to do, not what it actually did. *AbstractActions* (AA) refers to viewing real actions in terms of the abstract states they occur in. Essentially, this is the inverse concept to fixing the pot size when the agent takes an action. If it increases the size of its bets when the pot is too small, then it should view larger bets on too-small pots as smaller bets. Specifically, it can look at the disparity between the real pot and the pot in the abstract state and take this into account when translating the next observed action. For instance, if the opponent has committed 2 more chips to the pot in the real game than in the abstract game, then the agent will consider the next bet to be a bet of 2 chips more. This ensures that if in some state the abstract strategy says to take action a and the action b is taken in the real state, that later the agent will map action b to the abstract action a. Doing this makes sure that the retrospective view of an event matches the abstract actions the agent wanted to take during that event.

Another modification is *CallisCall* (CisC), which refers to maintaining call actions. There is a problem when the pot is smaller than the agent thinks it is, and instead of calling or checking it actually puts in a small bet to fix the size of the pot. Doing so is very dangerous because it allows the opponent to take another action. The fact that checking/calling affects the game tree quite

differently from a bet means that we should avoid, or at least be very careful about, turning calls into bets through translation. Thus, *CallisCall* simply ensures that whenever the abstract strategy tells the agent to call, it will perform a call action in the real game and not some other action.

The last modification is *ExtendCall* (EC). If the opponent makes a very small bet, it may be beneficial to view it as a check instead of a bet. For instance, if an agent bets 2 chips into a pot of 100, interpreting the bet as a pot bet of 100 greatly distorts the bet. By considering a check as a bet of 1, some small bets can be translated into checks. This can only be done in certain situations and requires some post processing of the end state. The first situation is if the bet is the first action, in which case mapping to a check has no effect on the game tree ($/b \rightarrow /c$). The second situation occurs when the bet is the second action and the first action was a check, which then gets mapped to no actions having occurred ($/cb \rightarrow /$). This, however, gets overridden by the last situation, which occurs when the action is the last raise action in a round. In this situation, it is mapped to a call and all the remaining actions in the round are ignored ($/...bc \rightarrow /...c$). This allows us to treat small bets as checks in some situations.

3.3.3 Results

We took an *fcpa* agent using the perfect recall 8s card abstraction (8.8.8.8) in the 500BB Texas Hold'em game as well as an *fcpa* agent in the 12-stack Leduc Hold'em game and applied combinations of these modifications to its translation. It should be noted that +-, *AllDown* and *naïvePA* use the same translation as the agent being tested, and accordingly the fixes affected their behavior as well. The unfixed translation was used by our agent in the 2007 competition, which ended up losing to the competitor BluffBot 2.0.[3] Table 3.4 shows the results of this experiment.

	Unfixed	CisC	EC	AA	EC,CisC	AA,CisC	AA,EC	All
+-	1061	1324	1540	1363	1630	1707	1450	1623
AllDown	1884	29	1276	2008	26	62	2021	50
naïvePA	12652	12652	12649	12252	12649	12252	12252	12252
BluffBot 2.0	270	6	101	–	-30	–	–	-105
Leduc BR	1634	710	1063	1634	1087	693	783	881

Table 3.4: Translation fixes effects in mb/h

Immediately we notice that almost all of these changes increased the exploitability of our agent against the +- player. However, we see the opposite when looking at the best response in the Leduc BR row. The most likely reason for this is that since the +- player uses the same translation as our agent, the fixes helped that agent as much (or more) than they helped Polaris.

It appears that the act of preserving calls (CisC) produced the largest improvement overall. The most dramatic change is against the AllDown player. This is likely because any time the AllDown player bet there is still the same chance of our agent folding, and a call by our agent would result

[3]The data for BluffBot is incomplete, and the existing data has a standard deviation around 125 mb/h. Unfortunately, we cannot obtain more data due to complications with the benchmark server.

in a raise, allowing the AllDown player to act again (and possibly raise if it had a good hand). This change also reduced the exploitability of the Leduc player by more than half, eliminating a large flaw in our strategies.

The $AbstractActions$ fix produced moderate improvements. Looking at the Leduc best response value, it is interesting to see that alone it has no effect, but when combined with one of the other fixes it provides some improvement. This is likely because the problem addressed by AA is ignored (by the best response) in lieu of the much larger problems that CisC and EC address.

The $ExtendCall$ fix produced mixed results. Although it generally provides an improvement over the unfixed translation, when combined with CisC it seems to produce worse results than CisC alone. This is seen mainly by the fact that the Leduc best response values for (EC,CisC) and All, 1087 and 881 mb/h respectively, get better when EC is removed, down to 710 and 693 mb/h. However, it is possible that the small stack sizes in the Leduc game mean that we do not encounter the ridiculous situations where a bet of 2 chips is considered to be a much larger amount. We decided to keep this fix until we could more soundly say whether it was good or bad.

Having all of these fixes implemented appears to provide a significant improvement in our translation. It reduces the exploitability of our agent by about half in the Leduc best response. Additionally, the data suggests that had these fixes been implemented for the 2007 competition, our agent may have defeated BluffBot.[3] Unfortunately, none of these fixes appear to have any significant effect on the naïvePA player. This is especially concerning, considering the fact that this player does not even have to look at its cards to be effective.

As a temporary defensive measure, we decided to add another betting option to help defend against the +- and naïvePA players. We wanted a betting option near the pot-allin boundary in order to minimize the effect the naïvePA player can have. For this reason we chose to add a 10-pot bet, with the added effect that it only increased the size of the (500BB) game by 68%. Adding this betting option is tantamount to throwing a giant boulder into the grand canyon to try to fill it. However, it is quite effective for handling the naïvePA player, reducing how much it exploits our agent from 12252 mb/h down to 451 mb/h. It also helps us against the +- player, reducing the value from 1623 mb/h to 324 mb/h. As this addition is only a band-aid for our translation, an actual solution to these exploitative strategies is discussed in Chapter 4.

3.4 Translation Concepts

When performing translation we generally query a strategy in an abstract game to decide what action to take. As the state of an abstract game may be different from the state of the real game, we have to choose how to act on these differences. This leads to two options, to adapt the abstract strategy to the real state (*real translation*), or to adapt the real state to the abstract strategy (*abstract translation*).

Abstract translation is the method Polaris generally uses. It attempts to view what is happening in the real game in terms of the abstract strategy it contains. For poker, this means that when making

bets it will modify the bet amounts in an attempt to push the pot size closer to the amount it thinks the pot should be. Additionally, it views real bets in terms of the abstract pot size instead of the real pot size. For example, if there are 16 chips in the real pot, 12 chips in the abstract pot and it observes a bet of 10 chips, it will consider this a bet of $10 + (16 - 12)/2 = 12$ chips into a pot of 12 chips (a 100% pot bet). This is exactly what the AbstractAction fix did as specified in Section 3.3.2. In essence, abstract translation always tries to move the real game state closer to an abstract state.

Real translation refers to viewing actions as occurring in the real game rather than the abstract game. This allows one to view the real situation more accurately, but it means that it is easier to get into situations that the abstract strategy does not model well. In poker, this approach refers to interpreting bets according to the real actions taken. For instance, if the pot size is 16 and someone makes a bet of 10, the agent will interpret this bet as a 63% pot bet. If an agent wants to make a pot bet, it will bet exactly the amount that is in the real pot (in a pot of size 16 it would bet 16). The important concept of this method is understanding the pot odds of an opponent bet, rather than the pot size.

It may seem that using real translation is better than abstract translation, but we expect it to perform worse. This is because using real translation allows an opponent to exploit the translation more. For instance, if the opponent is up-betting according to the +- exploitative strategy, then they can do so more effectively against an agent using real translation. For example, the opponent can up-bet multiple times in one hand when real translation is used, but only effectively up-bet once per hand against an agent using abstract translation. Since abstract translation views bets in terms of the abstract state, up-betting twice in a row is no more effective than up-betting on the second bet (since the abstract state is the same at the second bet). This is best explained with an example.

Assume we are at the start of a hand in the 500BB game, and the effective pot size is 4. The first up-bet would be of size $\lfloor \sqrt{4 * 998} \rfloor = 63$, and a call would make the pot $4 + 63 * 2 = 130$. The second up-bet (using real translation) would be $\lfloor \sqrt{130 * 935} \rfloor = 348$, and a call would make the pot $130 + 2 * 348 = 826$! Keep in mind that Polaris now thinks that two pot bets have occurred, making the pot size $4 + 2 * 4 = 12$ after the first bet and $12 + 2 * 12 = 36$ after the second bet. Using abstract translation, the pot discrepancy is calculated as the amount the opponent had contributed to the pot in the real state, $130/2 = 65$, minus the same amount in the abstract state, $12/2 = 6$. Thus, the second bet would be considered to be $65 - 6 = 59$ chips larger, whereas a pot bet would be the size of the abstract pot, in this case 12. This means the second up-bet could be as large as $\lfloor \sqrt{12 * 994} \rfloor = 109$, which is independent of the real pot size at that point. We then factor into account the pot discrepancy, meaning that the second up-bet would be $109 - 59 = 50$, making the pot of size $130 + 50 * 2 = 230$. This is equivalent to only up-betting the second bet, as that also results in the same size of $12 + 109 * 2 = 230$.

To test this hypothesis, we ran some experiments using both abstract and real translation. We used the +- player as described in Section 3.3.1 as well as the best response in the same Leduc game.

The results are shown in Table 3.5. As expected, real translation appears to be more exploitable than abstract translation. Thus, Polaris continues to use abstract translation to play.

Match	Real	Abstract
fcpta vs +-	682	324
fcpa vs +-	2243	1623
Leduc BR	1199	881

Table 3.5: Performance of real translation versus abstract translation in mb/h

3.5 Conclusion

In this chapter we discussed how the no-limit Polaris agents work. The two most important aspects of these agents are abstraction and translation. When choosing an abstraction, we must consider the trade-off between card abstraction and betting abstraction. We found that in no-limit, when considering card abstraction, it is most important to have as much current knowledge as possible. This means that we want as many buckets as possible on each round, forgetting any previous hand strength buckets. Doing so allows us to create better abstractions that use less space, giving us more space to allocate to the betting abstraction. Although it is very expensive to add additional betting options, it appears to provide significant improvements in the play of the agents.

The translation Polaris uses is somewhat complicated. When performing such translation, we must pay careful attention to exactly what we are doing. We want to make sure that we understand how translating action a into action b will affect the game tree, and avoid translations that result in poor behavior. Most importantly, we want to maintain internal consistency in that we never view a previous action as something different than what we originally intended. Even after fixing some of the problems that Polaris's translation had, it is still susceptible to exploitative techniques. Chapter 4 more formally discusses state translation and how to create more robust players.

Chapter 4

State Translation

4.1 Introduction

In this chapter we discuss the concept of state translation and how it affects the play of different strategies.[1] **State translation** refers to the process of translating a state in the full game to a state in an abstracted game. Specifically, the combination of a strategy in an abstract game and a translation function results in a strategy in the full game. Although the concepts described in this chapter apply to all extensive form games, we use poker as an example. In practice, an abstraction on chance nodes uses an explicit partition so that translation is just a table look-up. In poker, it is common to use a hand strength function to partition the two card starting hands into a fixed number of *buckets*, where the hands AA, KK and the other strongest hands are usually in the same bucket. However, the player action space is usually just restricted without an explicit partition being created. In poker, the actions may be restricted to *fcpa* without explicitly describing how a double-pot bet is handled. In this case, one must convert a real action history into a legal history in the abstract game in order to use the abstract strategy.

Using translation has several advantages. The first advantage is that we do not need to create a full partitioning of the set of real histories. In very large games, creating such a partition is not feasible and so not needing to create/store a partition is an important point. Another advantage is that after computing a strategy for the abstract game, we can create several players by implementing different types of translation methods. This means that one expensive equilibrium computation can result in a variety of strategies in the real game depending upon how the translation is performed. Finally, translation allows the player to take actions in the real game that are not legal actions in the abstract game. There are many situations in which knowledge about the real situation, which is lost in the abstract game, can influence what actions a player should take.

Throughout this chapter we will be using the extensive game notation defined in Chapter 2. In particular, we assume that we have an extensive game Γ as well as Γ' that is a loose abstraction of Γ. We will refer to the possible histories, H and H', and sets of legal actions, A and A', associated with

[1]Portions of this chapter appeared in [27]

these games, where H and A belong to Γ and H' and A' belong to Γ'. In Section 4.2 we discuss the current method of translation, first described by Gilpin and colleagues [9]. We also discuss ways to exploit a player using that method. Section 4.3 describes a new probabilistic translation method that produces more robust players. In Section 4.4 we illustrate both translation approaches in the context of poker. In Section 4.5 we show how an intelligent agent can learn the parameters of an opponent's translation function. Finally, the results of a variety of experiments are given in Section 4.6.

4.2 Hard Translation

4.2.1 Definitions

Hard translation refers to a translation function that is a mapping of real histories to abstract histories. Specifically,

Definition 8 *A **hard translation function** is a function on histories $T(h) \in H'$ where $h \in H$.*

Hard translation provides a partitioning of real-game histories that is sufficient to convert a loose abstraction into an explicit abstraction. A translation function can be used to define the partitioning $\alpha_i^{\mathbf{I}}$ where h, h' are in the same information set if and only if $T(h) = T(h')$.

The simplest way to implement such a translation function is to step through the history sequentially and convert each real action into a legal action in the abstract game.

Definition 9 *A **hard translation in-step function** is a function on histories and actions $t_{in}(h, a) \in A'(T(h))$ where $h \in H, a \in A(h)$.*

This allows us to recursively define the translation function as follows:

$$T((h, a)) = (T(h), t_{in}(h, a)). \tag{4.1}$$

An example of hard translation is as follows. Assume we are playing in a Texas Hold'em game and the betting sequence is $cr2c/cr6$ (c is a check/call, rX is a raise of size X and / denotes the end of a betting round). If the abstract game only allows bets of 0.5 or 1 times the pot, then the raise of 6 chips presents a problem since it is 0.75 times the pot.[2] The logical conclusion is to map the 6 chip raise to either 4 or 8.

$$cr2c/cr6 \begin{array}{l} \nearrow \; cr2c/cr4 \\ \searrow \; cr2c/cr8 \end{array} \tag{4.2}$$

It is important to note that there are other ways to define the translation function that give us more power in the translation. This type of step function ensures that the length of the abstract history returned is the same as that of the real history. This means that the action returned by the step function must have a similar affect on the game tree as the real action given. An example of

[2] The first call makes the pot size 4 chips. The raise of 2 is then a half pot bet, which ends up making the pot size 8 chips. The raise of 6 is then 0.75 of the pot.

this in poker is the act of translating a bet to a call or vice-versa. These two actions affect the game tree quite differently (for instance, translating a bet to a call could prematurely end a round) and using such a step function for translation would likely fail. Instead, there are situations in which we wish to translate a sequence of real actions to a sequence of abstract actions of different length. The easiest way to do this is to create exemption situations, in which we handle the translation in a special manner. Section 3.3.2 discussed how this was done for the Polaris agent, when we wanted to translate small bets into checks.

Ideally the step function would return the abstract action that is *most similar* to the real action given the real history. Thus, if we define a similarity metric then the step function can simply return the action with the highest similarity value.

Definition 10 *A **similarity metric** is a function on histories and actions* $S(h, a, a') \in \Re$ *where* $h \in H, a \in A(h), a' \in A'(T(h))$.

And the step function then becomes:

$$t_{in}(h, a) = \underset{a'}{\text{argmax}}(S(h, a, a')) \tag{4.3}$$

Looking at the previous example, we could define the similarity metric to be the ratio of the two bets. Thus, $S(h, r6, r4) = 4/6$ and $S(h, r6, r8) = 6/8$, where $h = \{cards\}cr2c/cr6$, and as $6/8 > 4/6$, the $r6$ action would be mapped to $r8$.

Finally, one last step is required. After a player has queried the abstract strategy for an action it must decide what action to take in the real game. The purpose of the out-step function is to translate the action in the abstract game into an action in the real game. This enables a player to take actions in the full game that are not legal actions in the abstract game.

Definition 11 *A **hard translation out-step function** is a function on histories and actions* $t_{out}(h, a') \in A(h)$ *where* $h \in H, a' \in A'(T(h))$.

Since most actions in the abstract game are legal actions in the real game, an agent could simply perform the action specified. However, there are a variety of situations in which one may want to do something else. An example of this in poker is when the pot size is different than we expect it to be. Suppose the size of the pot in the real game is 40 chips, whereas the size of the pot in the abstract game (after translation) is 36 chips. We know that the closer the pot size is in the real game to what the abstract model expects it to be, then the better the agent performs. Thus, if the abstract strategy says to make a pot bet (a bet of 36 chips), the agent will instead make a bet of 34 chips. If the opponent calls, the pot size in the real game becomes $2 * 34 + 40 = 108$, as the abstract game expects ($2 * 36 + 36 = 108$). Alternatively, we could argue that we want the pot ratio of the real bet to match the pot ratio of the abstract bet. Thus, the agent would then make a bet of 40 chips, since that is the real pot size. The translation concepts we use in poker were explained in detail in Section 3.4.

When defining these step functions it is important that we do so in a way that does not confuse the agent. Depending on how the equilibrium strategy was computed, it may have bad or no data in certain parts of the game tree. In particular, it may return bad data if asked what to do in a situation it believes has 0% chance of occurring. This situation is easily avoided by only performing actions the strategy says to perform. Although the agent would never perform an action the strategy says not to perform, it is possible that it would think that it did so. When interpreting its previous actions, it is important that it always maps the real action it performed back to the abstract action the strategy told it to play. Otherwise, it will start jumping around the abstract game tree and no longer play an equilibrium strategy within its own game. For instance, consider the history $cr2c/c$. The agent's strategy says to make a half pot bet (4), but $t_{out}(h, r4) = r6$, resulting in a raise of 6. The agent will then see the sequence $cr2c/cr6$. Since $r6$ maps to $r8$, it will translate the sequence to $cr2c/cr8$, different from the $cr2c/cr4$ it intended. We can ensure that this never happens by requiring the out-function to be an inverse of the in-function. If this is violated, then it can confuse the agent and result in poor play. We say that a translation function maintains **internal consistency** if the following holds:

$$t_{in}(h, t_{out}(h, a')) = a' \quad \forall\, h \in H, a' \in A'(T(h)) \tag{4.4}$$

Note that since t_{in} operates on a larger domain than t_{out} does, that the converse of this statement $(t_{out}(h, t_{in}(h, a)) = a)$ is not (and can never be) true $\forall\, h, a$.

We can see how maintaining internal consistency was part of the translation fixes implemented in Section 3.3.2. Primarily, the AbstractActions fix was designed to fix the internal inconsistency the agent had. The agent modified its actions to reflect a pot discrepancy, but it did not interpret previous actions in the same way. Thus, it was possible for it to attempt to perform an abstract action it would later translate into a different action. Similarly, the CallisCall fix also dealt with internal consistency. The agent would want to perform a call in the abstract game but instead performed a bet that it would later translate as a bet in the abstract game.

4.2.2 Weaknesses

Hard translation suffers from the fact that it is a many-to-one mapping. Looking at the $cr2c/cr6$ sequence again, we observe how this mapping is exploitable by considering the situations where $r6$ is mapped to both $r4$ and $r8$.

- $\left.\begin{matrix} cr2c/cr6 \\ cr2c/cr8 \end{matrix}\right\} \rightarrow cr2c/cr8$
- $\left.\begin{matrix} cr2c/cr4 \\ cr2c/cr6 \end{matrix}\right\} \rightarrow cr2c/cr4$

In the first situation, both $r6$ and $r8$ are mapped to $r8$. If we were going to make a bet of 8 and had bad cards (we are *bluffing*, trying to get our opponent to fold), we could bet 6 chips instead of 8. The action would have the same chance of causing the opponent to fold, but we would be risking fewer

chips if our opponent called. In the second situation, if we were going to make a bet of 4 and had good cards, we could bet 6 chips instead of 4. Here we increase the size of a pot we are likely to win without decreasing the probability of a call from our opponent, which we want because we believe we have a stronger hand.

The exploitative concept can be generalized to other situations. In particular, we know that for a given state, there exists some set of actions B that the agent would translate to the same abstract action c. It is not important to know the action c or how the agent will react, only that every action in the set B is treated equally. There are several ways to find such sets, varying from learning them (see Section 4.5) to simply knowing the similarity metric the player is using. The knowledge of how an agent translates the action space is much more dangerous than the knowledge of the card abstraction being used for two reasons.

The first reason is that agents can control what actions they take, whereas they have no control over what cards are dealt. This means that an agent can actively push the real game into situations where the opponent plays poorly. For this we would choose the action $a \in B$ that most confuses our opponent. This is likely the action with the lowest similarity to c, that is $argmin_a S(h, a, c)$. This results in the abstract state returned by translation being as far from the real state as possible. An example of this in poker is that making the largest pot bet possible (the largest bet such that the opponent interprets it as a pot bet) results in a situation where the real pot contains many more chips than the abstract pot. This results in the player having a poor concept of pot odds, especially when later faced with an all-in bet.

The second reason is that an agent need not know the opponent's strategy to exploit it. With card abstraction, an agent needs to know how the opponent plays within the information set to understand why it handles different situations in the real game poorly. With betting abstraction, knowing that an agent treats a set of actions the same is enough. Here the agent simply chooses the action $a \in B$ that maximizes its immediate utility, knowing that the opponent will not notice the effect. In poker, this can be done by modifying bet sizes according to hand strength. When making a bet that will be translated to a pot bet, the agent could instead make the largest bet possible that translates to a pot bet when it has a good hand and the smallest *pot bet* possible when it has a bad hand. This means that when the agent wins the pot is generally larger than expected and when it loses the pot is generally smaller. This is concept behind the +- player, described in Section 3.3.1.

It is possible that no opponent would understand the translation function enough in order to exploit it. However, it is possible to learn how an agent performs its translation. Assuming an agent is using hard translation, we need only learn which actions it responds to similarly and which it treats differently. We developed a method that can, with high accuracy and within 100 hands, estimate how an agent is performing hard translation. This method is described in detail in Section 4.5. In order to counter this method as well as the previously described types of exploitation, we desire a translation method that does not exhibit a *many-to-one* mapping that hard translation does. This leads to the

concept of **soft translation**.

4.3 Soft Translation

4.3.1 Definitions

A translation function is a **soft translation** function if it maps real histories to sets of weighted abstract histories. Specifically,

Definition 12 *A* ***soft translation function*** *is a function on histories* $T^s(h) \subseteq \Re \times H'$ *where* $h \in H$.

The concept is that maintaining several histories will provide us with a more accurate view of the world than just one. We can then sample the set of histories according to their weights to obtain an individual history that we act upon. We can also consider a hard translation function to be a special case of a soft translation function, in which one history has weight 1 and all other histories weight 0. Similar to hard translation, we can again define a step function that will help us implement this concept.

Definition 13 *A* ***soft translation in-step function*** *is a function on histories and actions* $t^s_{in}(h, a) \subseteq \Re \times A'(T^s(h))$, *where* $h \in H, a \in A(h)$.

We can use the similarity metric previously defined to determine the weights of actions returned. The step function then becomes

$$t^s_{in}(h, a) = \{(\alpha * S(h, a, a'), a')|a' \in A'(T(h))\} \tag{4.5}$$

where α is the normalizing constant. Soft translation makes it possible to differentiate states that were previously viewed as identical. While raises of 4 and 8 might both be mapped to themselves with 100% weight, a raise of 6 may be mapped to both states according to their normalized similarity values ($\{2/3, 3/4\}$ normalized is $\{0.47, 0.53\}$).

$$\begin{aligned}
cr2c/cr4 &\rightarrow \left\{ \begin{array}{ll} cr2c/cr4 & 1.0 \end{array} \right. \\
cr2c/cr6 &\rightarrow \left\{ \begin{array}{ll} cr2c/cr4 & 0.47 \\ cr2c/cr8 & 0.53 \end{array} \right. \\
cr2c/cr8 &\rightarrow \left\{ \begin{array}{ll} cr2c/cr8 & 1.0 \end{array} \right.
\end{aligned}$$

Again we can step through the action history, except now we convert every real action into a weighted set of abstract actions. By weighting these actions by their (normalized) similarity values, we obtain a more accurate analog of what actually happened in the real game. When performing such a translation, we find that the number of (non-zero) weighted histories grows exponentially in the number of real actions. One way to avoid this is to sample the actions returned by the step function according to their weights instead of storing all of them. This does not affect the final distribution and allows us to perform translation while maintaining only one abstract history. In this manner we can view soft translation as a non-deterministic version of hard translation.

The out-step function need not be defined differently for soft translation. This is because sampling the set of histories and the abstract strategy still results in one abstract action to perform. This action can then be translated out using the same function as described for hard translation. However, we again need to be careful that we do not confuse the player. If $T(h)$ returns $\{b, b'\}$ and the player samples from b to choose the action a, then it may not want to sample from the resultant history $h' = b'a$ later on since b' may suggest to never take action a. This would result in the player querying a bad part of the game tree, which can result in unpredictable behavior. The way to fix this is different depending on whether the player is sampling the set of weighted actions throughout translation or maintaining the full (exponentially increasing) set. From a game theoretical standpoint, both of these methods result in the same distribution over selected actions.

If the player is using the exponential version of soft translation, then it needs to update the weights of the histories by the probability it would have taken such an action. For instance, if the strategy for b is to perform a 70% of the time and a' 30% of the time, and the strategy for b' is a' 100% of the time, then the weight of the resultant history $h = ba$ is multiplied by 0.7 (the probability it would have performed a given b) and the weight of the resultant history $h' = b'a$ by 0 (the probability it would have performed a given b').

If the player samples from the action sets it received, we need to enforce that, within one game, the step function returns the same abstract history given the same input. In other words, given $h \in H$, we desire that the $h' \in H'$ returned by translation is the same within one hand. Assuming the sampling process is governed by a random number generator, we can seed the random number generator with a game ID (or some hash of said ID). This ensures that the history h will be translated in the same way given the same ID, but perhaps differently should it arise in another hand.

4.3.2 Effects

The primary focus of soft translation is to allow the player to distinguish between two similar states. In particular, soft translation gives us the guarantee that two actions $a, b \in A(h), a \neq b$ elicit different responses from the player (assuming that the player reacts differently to different actions in the abstract game). The weights of the states also give the player a better understanding of every state, meaning that there are fewer states in which the player has a poor understanding of the real situation.

The concept of maintaining several histories perhaps makes more sense in situations where the opponent's action is hidden. In attempting to model such a situation, simply assuming the most likely event occurred would result in the player being unprepared when this assumption is incorrect. Instead, one can model the situation by weighting different events according to the probability that they occurred. Our situation differs in that we *know* what our opponent did, but we do not *understand* what that action means. Just as we would describe a motorcycle as a mixture of a bicycle and a car, describing an unknown situation as a mixture of known situations can more accurately describe the

real situation.

Unfortunately, soft translation gives us no guarantee that our agent will perform the correct action. It is possible that none of the strategies for the returned histories contain the correct response since they simply cannot model the situation accurately enough. However, when dealing with actions that do not exist in the abstract game to begin with, all guarantees of optimality are lost and we are stuck using methods without useful bounds on their worst case scenarios.

4.3.3 Dominated Actions

The concept of considering multiple possible histories provides us with another interesting concept. In games there is the concept of a dominated action, an action a in which one is guaranteed a lower utility in all situations than some other action a'. We say that a' *dominates* a at a particular information set if the utility of performing a' is greater than the utility of performing a for all possible states within the information set and for all possible strategies the opponent could play. As performing dominated actions is always a mistake, we seek strategies that do not perform them. In fact, in full information games a strategy that never takes a dominated action is actually an equilibrium. This is because equilibria in full information games do not need to hide information, and thus every full information game has an equilibrium that does not mix over its action space (it is a pure strategy) [24, pages 163-171]. In full information games every non-dominated action is taken with some probability in some equilibrium, and it is safe to switch between equilibria (unlike in partial information games, see Chapter 5). Thus, a combination of non-dominated actions results in an equilibrium strategy.

The concept of non-dominated actions can be extended to soft translation. In an abstract state, we consider any action to be dominated if the strategy for that state says to perform it with 0% probability (or $< \epsilon\%$ probability for small ϵ) and it is a legal action to perform. Now consider the weighted set of abstract histories returned by a soft translation function. Define B to be the set of histories with weight $> \epsilon\%$. By querying the abstract strategy for each history $h' \in B$, we can determine which actions are dominated in each history $D_{h'} \subseteq A'(h')$. We can then define the union of these actions $D^s = \cup_{h' \in B} D_{h'}$ to be the set of pseudo-dominated actions. This set of actions is then removed from the legal set of actions for every history in B, and we ensure that the player does not take an action that is dominated in one of the histories it is partially in.

Removing the set of pseudo-dominated actions from the set of legal actions is a dangerous concept to implement. The intuition comes from a strong underlying concept of not performing dominated actions, but we have no theoretical guarantees that what we are doing will have a positive impact. There are two obvious flaws to using this idea. The first is that the behavior of a player using this concept is no longer smooth in that a slight alteration in actions can result in a significant change in play. For example, if in poker we made a bet of 4 chips into a pot of 4 chips the player would interpret this as a pot bet with weight 1, however if we bet 5 chips it would mix between

a pot bet and the next highest bet b. Thus, any dominated actions from the strategy for b would be invalidated even if b is a bet of $1,000$. Thus, b could suggest that all actions are dominated by folding, and the player would then fold even if the player would call a pot bet. The second flaw is that it is possible to invalidate all legal actions. If every legal action is dominated in one of the returned histories, then we suddenly have no legal actions to choose from. For instance, if a raise is interpreted as a pot bet then the fold action is dominated, whereas if it is interpreted as b everything but fold is dominated, then no actions remain. Although these issues cannot be immediately addressed in an intelligent manner, we can still test this concept by performing some default action (check/call) whenever translation fails and observe the results.

4.4 Application to Poker

4.4.1 Hard Translation

With the translation methods, real game and abstract game defined, all that is needed to implement hard translation is a similarity metric and an out-step function. Recall that the most common betting abstraction used in the no-limit game allows each player to fold, call, bet pot, or go all-in (fcpa). This means that every bet must be translated to one of these four actions. The metric used by several of the competitors in the AAAI no-limit poker competitions was described by Gilpin *et al.*[9] and will be formalized here.

Definition 14 *The **geometric similarity** of a real action a and a legal action a' in the abstract game is as follows, where b, b' are the respective bet sizes associated with a, a'.*

$$S(h,a,a') = \begin{cases} b/b' & \text{if } b < b' \\ b'/b & \text{otherwise} \end{cases} \tag{4.6}$$

This is the metric we use in our translation function.

To define the out-step function, we need to map the legal abstract actions to real bet amounts. We will do this in two ways, using both *real* and *abstract* translation as defined in Section 3.4. As long as we apply whichever concept we are using to both the out and in translation step functions we can maintain internal consistency.

Knowing the similarity metric we can immediately see how a player using the current translation method and the $fcpa$ abstraction would interpret certain bets. For instance, if p is the number of chips associated with a pot bet and a is the number of chips associated with an all-in bet, then we can find the bet b that is 50% between p and a. This occurs at $\frac{p}{b} = \frac{b}{a} \Rightarrow b^2 = pa$. Thus, we know that any bet larger than $\sqrt{pa}$ will be interpreted as all-in, and any bet smaller than that will be interpreted as a pot bet. Similarly, if we consider a *check* to be a bet of 1, then $\sqrt{p}$ is the border that determines whether a bet is considered a pot bet or a check/call.[3] This means that any amount from $\sqrt{p}$ to $\sqrt{pa}$

[3]Since calling affects the game tree differently than a bet, we can only translate real bets into check/calls in certain situations.

will be interpreted as a pot bet, and we can choose to use whichever one will benefit us the most knowing such a player cannot tell the difference.

4.4.2 Soft Translation

A slightly different metric is used for our soft translation function. Looking at the previous metric, we see that every action will always have a non-zero similarity value. However, since the weights of all actions are important in soft translation, we desire that when the similarity value of one action is 1 that the values of all other actions are 0. Because the different bet sizes lay on the number line, we only need to consider the closest legal bets larger and smaller than the actual bet (all other actions are given weight 0). If $b_1 < b < b_2$ where b is the real bet associated with a and b_1 and b_2 are the bets of the two closest legal abstract actions a_1, a_2, then the metrics are as follows.

$$S(h, a, a_1) = \frac{b_1/b - b_1/b_2}{1 - b_1/b_2} \tag{4.7}$$

$$S(h, a, a_2) = \frac{b/b_2 - b_1/b_2}{1 - b_1/b_2} \tag{4.8}$$

Thus, we have that the metric $S(h, a, a_1) = 1$ when $a = a_1$ and $S(h, a, a_1) = 0$ when $a = a_2$, as desired. An important aspect of this property is that if the original history being translated is a legal history in the abstract game, then soft translation will return this history with weight 1.

We also see how the previous exploitative methods no longer work. This is because bets near $\sqrt{p}$ will return (0.5-c,0.5-p) and bets near $\sqrt{pa}$ will return (0.5-p,0.5-a). Thus, a bet of $\sqrt{p} + 1$ and $\sqrt{pa} - 1$ will be treated quite differently. Additionally, bets near $\sqrt{pa}$ will be treated similarly, so attempting to manipulate bets near that border will be less obviously effective.

4.5 Boundary Exploration

Although we know that hard translation can lead to exploitable behavior, we still need to understand how the translation is being implemented in order to exploit it. If we assume an agent is using hard translation with a betting abstraction of type *fcpa*, then we need to find the boundary between p and a that determines whether a bet is translated down to p or up to a. Finding this location will specify the range of bets that are interpreted as pot and all-in bets, and thus allow us to exploit the translation.

4.5.1 Problem Definition

Assume the agent has two legal bet amounts a and b and any bet c between them, $a < c < b$, must be translated to either a or b. There then exists a true boundary point θ^* such that:

$$c = \begin{cases} a & \text{if } c \leq \theta^* \\ b & \text{if } c > \theta^* \end{cases} \tag{4.9}$$

Our goal is to find θ^* as quickly and accurately as possible. This is done by observing the agent's responses to various actions. We assume that the set of responses come from two independent distributions, one associated with a and one with b. If we have an estimated boundary point θ, then we can assign all responses to bets in $[a, \theta]$ to the a distribution and all responses to bets in $(\theta, b]$ to the b distribution. This allows us to evaluate the likelihood that θ is the correct boundary point.

First we discretize the betting space such that we have n possible bets to investigate, all of which fall in $[a, b]$. Although we could simply distribute the n bets uniformly over $[a, b]$, this need not be the case. If we believe the boundary to be at one of several points, we can directly investigate those locations. For instance, in poker two common boundary points are the arithmetic mean, $\frac{a+b}{2}$, and the geometric mean, $\sqrt{ab}$. If we know an agent is using one of these two points, then we need not investigate other locations.

In order to estimate the boundary location, we store a set of observations of the agent. For each of the possible bet amounts, we store how often the agent folds, calls and raises in response to the bet. This provides us with a matrix S that is our observation set:

$$S = \begin{bmatrix} F_1 \ldots F_n \\ C_1 \ldots C_n \\ R_1 \ldots R_n \end{bmatrix} \tag{4.10}$$

where F_i the number of observed instances of a fold in response to the i bet (and similarly for C_i, R_i). Given an estimated boundary θ we can define observation sets for the two assumed distributions. Let (F_a, C_a, R_a) be the observations associated with the a distribution and (F_b, C_b, R_b) be the observations associated with the b distribution. We then have $F_a = \sum_{i=1}^{\theta} F_i$ and $F_b = \sum_{i=\theta+1}^{n} F_i$ and similarly for the others.

Once we have the observed distributions, we can evaluate their likelihood (and thus the confidence that θ is the correct boundary point). Let $V(\theta, S) \in [0, 1]$ be an evaluation function that assigns a confidence value for the boundary point θ given S. Similarly, let $V(S) = max_{\theta} V(\theta, S)$ be the confidence value of the observation set S. The concept is that we continue to obtain data until such a point that $V(S)$ exceeds some confidence threshold. At that point we believe to have a good estimate of the boundary point.

When calculating $V(S)$ we do not want several θ to have high confidence values. It does not make sense to be highly confident that several boundary points are correct when only one can be. Thus we assume that $V(\theta, S)$ is normalized over the possible θ. Normalizing the confidence values ensures that we consider the confidence of different points relative to each other. This allows us to use metrics that may assign high confidence levels to multiple points or, in our case, assign low confidence values to all points. The next step is to define this evaluation function.

4.5.2 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is an optimization method where the assumed distributions are assigned values to maximize the probability of the observed data. First, we define f_a, c_a

and $r_a = 1 - f_a - c_a$ to be the probability of observing a fold, call and raise given that we pulled from the a distribution. We can then state the probability of observing the given data as follows:

$$P_a(S|\theta) = f_a^{F_a} * c_a^{C_a} * r_a^{R_a} \tag{4.11}$$

Our goal is then to find f_a, c_a and r_a that maximize $P_a(S|\theta)$. These values are found to be the empirical probability of the data.

$$f_a = \frac{F_a}{F_a + C_a + R_a} \tag{4.12}$$

$$c_a = \frac{C_a}{F_a + C_a + R_a} \tag{4.13}$$

$$r_a = \frac{R_a}{F_a + C_a + R_a} \tag{4.14}$$

The probability of observing the full data set is then the product of the probabilities of observing the a and b distributions.

$$P(S|\theta) = P_a(S|\theta) * P_b(S|\theta) = f_a^{F_a} * c_a^{C_a} * r_a^{R_a} * f_b^{F_b} * c_b^{C_b} * r_b^{R_b} \tag{4.15}$$

The evaluation function is then the normalized probability of observing the data.

$$V(\theta, S) = \alpha * P(S|\theta) \tag{4.16}$$

where α is the normalizing constant.

4.5.3 Estimation Algorithm

Given an observation set S we can assign a value to how confident we are in estimating the boundary point. If the confidence value is not high enough, we can obtain more data in the hope of obtaining a more confident bound. Unlike many problems, we have direct control over the additional data we obtain. Specifically, we can choose which bet option could benefit most from an extra data point. Thus, we desire a method for determining which action is most likely to increase the confidence value of the observation set by the largest amount. The simplest way to do so is to directly estimate the possible future confidence values of the observation set. Given the current data set S, we can estimate the probability of obtaining a fold/call/raise for a particular action. We then assume that we received such a response, update S, and calculate the new $V(S)$. The action to take is the action that results in the highest estimated $V(S)$. Algorithm 1 describes this process in detail.

This algorithm tells us what action to perform given an observation set S. All we have to do to find the boundary point is to repeatedly query this algorithm until we obtain a sufficiently high confidence value. Algorithm 2 describes this process.

4.5.4 Results

We implemented the EstimateBoundary algorithm (Algorithm 2) and tested it against a version of Polaris. Specifically, we used an agent with the 5.5.5.5 card abstraction and the *fcpa* betting

```
ChooseAction(S):
Input: Observation set S of size [m, n]
Output: pos
maxval = 0;
pos = 0;
for i = 1 to n do
    val = 0;
    total = 0;
    for j = 1 to m do
        total = total + S[j, i];
        S' = S;
        S'[j, i] + +;
        val = val + S[j, i] * V(S');
    end
    val = val/total;
    if val > maxval then
        maxval = val;
        pos = i;
    end
end
return pos;
```

Algorithm 1: Obtain the next action to perform

```
EstimateBoundary(actions, γ):
Input: Array of n possible actions, confidence halting criterion γ
Output: pos
S ← nullmatrix;
for i = 1 to n do
    Perform action[i], update S according to response
end
while V(S) < γ do
    Perform action[ChooseBet(S)], update S according to response
end
return argmax_i V(i, S);
```

Algorithm 2: Explores the action space to find a split location

abstraction, where we modified the translation function to use different boundary locations. We chose to use 20 betting amounts uniformly distributed in $[p, a]$, the range between pot and all-in. We set the confidence criterion γ to be 0.99 and set a hard cap of 500 hands. Although computer agents can play millions of hands without difficulty, the matches run in the AAAI no-limit competition only run for a couple thousand hands each. Thus, speed is very important and we decided that if the algorithm could not converge to a boundary point in 500 hands then it was learning too slowly. It was determined a failure any time the algorithm converged to the wrong boundary point or hit the 500 hand mark.

The algorithm was run for 100 trials on each of five different boundary points. We recorded the average iteration of convergence as well as the error rate and the average error (how far the estimated boundary point was from the actual boundary point). A boundary point of 0.1 refers to the boundary being at $0.1 * (a - p) + p$. Table 4.1 shows the results of the experiment.

Boundary Point	0.1	0.3	0.5	0.7	0.9
Average Iteration	66.86	79.75	93.23	99.56	114.60
Error Rate	0%	0%	1%	1%	4%
Average Error	0	0	0.0035	0.0050	0.0160

Table 4.1: Effectiveness of EstimateBoundary at predicting a Polaris agent's boundary point

Table 4.1 shows that convergence is reached quite quickly and with high accuracy. Not shown in this table is the fact that most of the failures came from the algorithm converging to the wrong location. Thus, we could raise the confidence criterion to lower the error rate for a slight performance hit in the convergence time. Regardless, this shows that a hard translation function can be exploited by an unknowledgeable opponent with a good learning algorithm.

4.6 Results

We constructed a series of experiments to compare the performance of hard and soft translation. We created three no-limit agents for each no-limit variant we used. Within each variant, the three agents use the same solution to the abstracted game, but one uses hard translation, one uses soft translation, and one uses soft translation with dominated actions removed (ND). The first variant used is the *fcpa* betting abstraction of the full no-limit 500BB game. The second variant uses the *fchpta* betting abstraction in the 100BB game. Both of these variants use the ir169.64.8.8 card abstraction. The final variant is a Leduc Hold'em game with stacks of size 12. For the Leduc game, a variety of betting abstractions were used, namely the power set of the *hpd* betting options, with *fca* always allowed. For the Leduc agents, we can directly calculate the value of the best response to their strategies. Since the other games are too large to calculate this value, the agents in the first two variants are played against a set of opponents designed to test their translation abilities. Additionally, we reinvestigated the effects of real translation versus abstract translation (see Section 3.4).

4.6.1 Opponents

Since the full no-limit poker game is too large to compute a proper best response to our agents, we created a variety of different opponents to test the agents. Some of the opponents were designed to exploit hard translation and others simply play using a strategy created with a different betting abstraction. Note that in this section, a *large* pot bet or a *small* pot bet refers to making the largest or smallest possible bet that our opponent will interpret as a pot bet. This notation will also be used when referencing bet amounts other than a pot bet.

The first set of opponents that were created were designed to exploit hard translation. This exploitation is done by controlling the size of the pot in a way that is invisible to a player using hard translation. Knowing, for instance, the range of bets that the player interprets as a pot bet allows us to make larger or smaller pot bets and thus control the size of the pot. Controlling the pot size allows us to exploit the player in two ways. The two players based on these exploitative methods are the +- and *naïvePA* players. These players are the same as the ones described in Section 3.3.1.

Several other players were constructed using the same concepts as the +- player. The -+ players works the same way except it reverses the type of bet it makes based upon its hand. We expect that reversing the +- strategy will have the opposite effect on the amount of money won against the player. Two other opponents, *+1-1* and *-1+1*, are variations on these techniques. When making a bet, these players will instead bet 1 chip more or less depending on the strength of their hand. These players were mainly created to test how robust the soft-ND translation is. By varying their bets by only one chip, they cause pseudo-dominated actions to be removed without greatly changing their bets.

Lastly we have two opponents that do not use exploitative techniques. These opponents are simply equilibrium solutions to different betting abstractions. This means that they will take actions that need to be translated by the player, but these actions are not designed to take advantage of how the player's translation method works. *fc75pa* and *fc125pa* bet 75% and 125% of the pot instead of 100% of the pot, respectively. These players do not use the same strategy as their opponent, but rather the solution to their own abstracted games.

In summary, *naïvePA*, +- and *+1-1* are all designed to exploit the normal translation method to different degrees. The inverse players, -+ and *-1+1*, are weak agents that manage to hurt themselves by exploiting the translation in the wrong direction. *+1-1* and *-1+1* were designed to test the robustness of soft-ND translation by betting just off of the expected amounts. Lastly, *fc75pa* and *fc125pa* are designed to see how well the methods handle bets that are non-exploitive but also not part of the abstraction. All of these agents use hard translation when playing.

4.6.2 Abstract Translation

This section shows results using abstract translation. Table 4.2 shows the results for the 500BB game using the *fcpa* betting abstraction. It is important to note that a player that folds every hand loses at

most 750 mb/h. Additionally, in last year's AAAI no-limit competition, first place beat second place by 109 mb/h.

	Hard	Soft	Soft ND
naïvePA	15458	5696	-6114
+-	1053	554	-95
-+	-1666	-401	-2233
+1-1	30	21	19
-1+1	-26	-9	-12
fc75pa	14	30	26
fc125pa	-30	0	14

Table 4.2: Translation results for 500BB players in mb/h using abstract translation

Table 4.2 shows that naïvePA beats the player using hard translation by 15458 mb/h. This is amazing considering that naïvePA does not look at the cards it is dealt. We also see that naïvePA beats the player using soft translation by 5696, a significantly smaller amount but still positive. The naïvePA player is still effective against soft translation because the second bet it makes (the small all-in bet) is a relatively small amount due to the effects of using abstract translation. Additionally, its exploitative technique is still working 25% of the time. Each bet is interpreted as a pot bet roughly 50% of the time and as an all-in bet 50% of the time. This means that 25% of the time it works correctly (pot followed by all-in), 25% it fails critically (pot followed by pot), and 50% it fails safely (all-in on the first bet, which generally results in the player folding and naïvePA winning the blinds). Since the cost for its technique failing is relatively low, it still profits. Later we will see that this is not the case when there is an additional bet between pot and all-in or when using real translation. Finally, removing dominated actions results in naïvePA losing by 6114 mb/h. This is largely attributed to the *fold* action being disallowed when the agent has a decent hand, resulting in the agent calling the bets down more often and thus increasing the proportion of hands where naïvePA's technique fails critically.

Similarly, the +- player's winnings are reduced by soft translation. It beats the agent using hard translation by 1053 mb/h, which is reduced to 554 mb/h by soft translation. Conversely, we see that soft translation does not beat the inverse players by as much as hard translation. This makes sense, since the goal of soft translation is to reduce the effect of this type of exploitation. Thus, if it defends against an exploitive method then it likely exploits the inverse method less. Again, we see that soft-ND translation provides a significant improvement over soft translation in both situations.

When playing against the +1-1 and -1+1 players, soft translation had a similar effect as against the +- and -+ players. It reduced the exploitative technique slightly without completely defending against it. However, we notice that soft-ND performed very closely to soft translation. This means that the possible complication of removing actions does not appear to be an issue in general.

Against both the fc75pa and fc125pa players soft translation performed slightly worse. Soft-ND performed similarly to soft translation or slightly worse. It appears this method does not have the

same effect on these players as it does on the exploitative ones. This may be because the exploitative methods are using the same abstraction as the agent being played against, whereas the fc75pa and fc125pa players are using a different abstraction.

	Hard	Soft	Soft ND
naïvePA	944	-591	-3712
+-	282	48	49
-+	-202	-8	-136
+1-1	19	13	14
-1+1	-46	-19	46
fc75pa	-40	-7	1
fc125pa	-138	-96	43

Table 4.3: Translation results for 100BB players in mb/h using abstract translation

Table 4.3 shows the results for the 100BB game using the *fchpta* betting abstraction. The data shows mostly the same results as the 500BB game, with two notable exceptions. First, soft translation turns a win for naïvePA into a loss. This occurs because the cost for naïvePA's technique failing is much higher due to the extra ten-pot bet option. Second, we see that soft-ND translation results in larger losses against the -1+1 and fc125pa players.

	Hard-P1	Soft-P1	SoftND-P1	Hard-P2	Soft-P2	SoftND-P2
fchpda	0.26	0.18	0.39	0.44	0.36	0.36
fcpda	1.01	0.84	0.86	0.61	0.59	0.59
fchda	0.49	0.28	0.28	0.54	0.42	2.22
fchpa	0.70	0.28	0.56	0.47	0.38	0.38
fcha	0.64	0.26	1.28	0.89	0.49	2.84
fcpa	1.14	0.90	1.04	0.62	0.59	0.62
fcda	0.54	0.53	0.98	0.61	0.63	0.64
fca	1.10	0.54	1.12	1.28	0.75	1.26

Table 4.4: Exploitability of various 12-stack Leduc Hold'em players in sb/h using abstract translation

Table 4.4 shows the results for the 12-stack Leduc Hold'em game. Hard-P1 shows the exploitability of a player using hard translation in mb/h by a knowledgeable opponent sitting in position 1 (and similarly for the other columns). Each row shows the results for a player using a different betting abstraction. It is important to note that in this game each player antes 1 chip instead of posting blinds, and thus the always-fold player loses 1 sb/h.

Looking at the table, we see that the exploitability of a player using soft translation is almost always smaller than the exploitability of a player using hard translation. The one exception is position 2 against the fcda player, in which soft translation is exploitable by 0.63 sb/h instead of the 0.61 sb/h exploitation of hard translation. Looking at the effects of using soft-ND translation, the results are mixed. It many cases it is less exploitable than hard translation, though rarely does it perform better than soft translation. Additionally, some times it creates very poor players, in particular the fcha player in which it increases the original exploitability of 0.64 and 0.89 to 1.28 and 2.84. It

seems that an opponent who knows exactly how the soft-ND translation fails can exploit it for a large amount.

4.6.3 Real Translation

In this section we show results of using real translation. Table 4.5 shows the results for the players in the 500BB game using the *fcpa* betting abstraction. These results have several differences from the results using abstract translation (Table 4.2). First, the naïvePA player wins only 13506 sb/h instead of 15458. Additionally, it loses money when soft translation is used. This is because real translation forces the second bet to be at the real boundary between pot and all-in, which is a larger amount than when using abstract translation. Thus, when the technique fails the naïvePA player loses many more chips than when using abstract translation. However, this same reason causes the player to be more exploitable to the +- player and its variants. The amount won by the +- player increases to 2711 sb/h from 1053, a significant increase. Aside from these changes, the trends in the results are mostly the same as before. The same can be said for the 100BB game (using the *fchpta* betting abstraction), shown in Table 4.6.

	Hard	Soft	Soft ND
naïvePA	13506	-2753	-12530
+-	2711	841	682
-+	-2582	-582	-1979
+1-1	74	59	59
-1+1	-69	-42	-43
fc75pa	7	34	21
fc125pa	-5	-8	12

Table 4.5: Translation results for 500BB players in mb/h using real translation

	Hard	Soft	Soft ND
naïvePA	517	-1569	-3929
+-	466	61	23
-+	-472	-70	-205
+1-1	62	47	24
-1+1	-63	-8	55
fc75pa	-12	-6	-8
fc125pa	-126	-64	70

Table 4.6: Translation results for 100BB players in mb/h using real translation

Table 4.7 shows the results for the 12-stack Leduc Hold'em game. We see that the exploitability of soft translation is smaller than the exploitability of hard translation in every situation. If we compare the values in this table to those in Table 4.4, we see that the exploitability using real translation is higher than when using abstract translation in almost every situation. This confirms that abstract translation appears to be more robust than real translation.

	Hard-P1	Soft-P1	SoftND-P1	Hard-P2	Soft-P2	SoftND-P2
fchpda	0.41	0.23	0.42	0.50	0.41	0.41
fcpda	1.18	0.84	1.56	1.23	1.12	1.12
fchda	0.61	0.25	0.25	0.57	0.46	1.96
fchpa	0.76	0.38	0.61	0.52	0.43	0.43
fcha	0.86	0.36	1.46	1.06	0.61	2.84
fcpa	1.44	0.92	1.95	1.04	0.93	0.93
fcda	0.84	0.51	1.16	1.39	1.13	1.20
fca	1.19	0.59	1.12	1.61	0.88	1.36

Table 4.7: Exploitability of various 12-stack Leduc Hold'em players in mb/h using real translation

4.7 Conclusion

In this chapter we formally described the methods of translation used to handle extensive games with large action sets. Additionally, we looked at the current method of translation, described why it could result in an exploitable agent and showed examples of how this can be done in poker. We also described a new probabilistic translation method that helps counter these exploitative techniques. This new method greatly reduced how exploitable the agent was to these techniques. Additionally, we experimented with removing pseudo-dominated actions in soft translation, resulting in agents that are more robust to simple exploitative techniques but more exploitable against a fully knowledgeable opponent. Unfortunately, our data also showed that an agent using soft translation could suffer a minor performance loss when playing non-exploitative opponents that play using a different action abstraction. It is possible that further development of this technique can reduce or reverse this performance loss. In addition to exploring the effects of soft translation, we revisited the effects of real versus abstract translation. The data showed that even when using soft translation, abstract translation appears to perform better than real translation.

Unfortunately, as the real game space is too large to calculate exactly how exploitable these players are, we cannot say anything definitive about how well these translation methods perform in full no-limit Texas Hold'em. However, we do know that soft translation produces more robust players in Leduc Hold'em, and we could assume a similar situation would hold in larger games. Regardless, naïvePA does not even have to consider its own cards in order to win a significant amount against a player using hard translation. As no-limit poker programs progress toward being competitive against world-class human players, such an obvious flaw cannot exist.

Chapter 5

Strategy Switching

5.1 Introduction

The concept of strategy switching is to change the strategy being followed in the middle of a game. There are several reasons for doing this. The first is that a strategy may only be defined for a certain portion of the game, and thus another strategy is needed for other parts of the game. This occurs when a game is split into separate parts in order to be solved. For instance, we could split poker into a preflop model consisting of the first 3 rounds and a postflop model consisting of the last 3 rounds. By computing a strategy within each of these sub-games and switching between these strategies according to what round it is, we obtain a strategy for the full game. This is how the Opti [1] agent worked.

Another reason for switching between strategies is to use different abstractions. If the underlying abstractions used by the strategies are different, then certain situations in the full game would more closely map to one abstraction or another and we could use the strategy associated with the abstraction that best captures the real situation. For instance, if we have two poker strategies, one which uses the $fchpa$ betting abstraction and one which uses $fcpdta$, then it seems logical that we would want to use the $fchpa$ strategy when a half pot bet is observed in the full game. Similarly, if a double pot bet is observed we would want to use the $fcpdta$ strategy. In this way we can create an agent that understands both half pot and double pot bets, assuming that both sized bets do not occur.

Switching between strategies is not a new concept and is generally considered to be a bad idea. This is because in imperfect information games switching between two equilibrium strategies does not necessarily result in an equilibrium strategy. However, when dealing with abstraction we find that strategy switching can provide some advantages. Section 5.2 describes why this concept can be implemented safely in perfect information games. Section 5.3 shows why this concept is not safe in games of imperfect information. Section 5.4 describes the process of creating a set of strategies in Leduc Hold'em that cover all *important* situations in the game. Finally, Section 5.5 shows the results of some experiments involving strategy switching.

5.2 Perfect Information Games

In a game of perfect information, it is possible to switch between several carefully chosen equilibrium strategies in a way that results in playing an equilibrium strategy. Because we have perfect information in the game, the actions taken later in a game are seemingly independent from the actions taken earlier in the game. This leads to the concept of subgame perfect equilibrium [22, pages 162-176]. A strategy is a **subgame perfect equilibrium** if it is an equilibrium strategy in every subgame of the full game. In this situation, such a strategy can be swapped in at any information set in the game and produce equilibrium play from that point on. Thus, switching between subgame perfect equilibria within a game results in playing an equilibrium strategy.

The most common method for obtaining a subgame perfect equilibrium is using Minimax [24, pages 163-171]. Minimax works by using backwards induction from the terminal nodes of the game tree. It computes the optimal move at the bottom of the game tree and stores the value of taking that action. It then backs up the tree, calculating the optimal move at every level since the value of the child states are known. Once the algorithm hits the root node it is finished and describes an equilibrium solution for the game. This results in a subgame perfect equilibrium because the algorithm considers only the ancestors of a state when calculating the strategy for that state, resulting in an equilibrium strategy for every branch of the tree and thus a subgame perfect equilibrium. This means that in perfect information games we can efficiently find equilibrium strategies that can safely be switched between without losing any guarantees of optimality.

5.3 Imperfect Information Games

In games of imperfect information, subgame perfect equilibria generally do not exist. This is because most equilibrium solutions must mix their actions and how they play later in the game is dependent upon how they mix their actions earlier in the game because their opponents' beliefs are affected by these previous actions. However, this in itself does not show that switching between two equilibrium strategies within a game will result in poor play. For this we turn to an example in Kuhn poker, a game described in Section 2.1.2.

In Kuhn poker the equilibria for the first player take the form of $\alpha = \gamma/3$, $\beta = (1 + \gamma)/3$ for any $\gamma \in [0, 1]$. α represents the probability the player bets with a Jack, β represents the probability the player calls a bet with a Queen, and γ represents the probability the player bets with a King. All other situations are dominated, specifically one never bets with a Queen, always calls a bet with a King, and never calls a bet with a Jack. The second player should always bet with a King since the game ends if the player checks. The equilibrium for the second player is to both bet with a Jack and call a bet with a Queen $1/3$ of the time. When playing an equilibrium, the first player is expected to lose $1/18$ antes per game.

Within Kuhn poker, we can consider switching between two equilibrium strategies for the first

player. We use strategy $\gamma = 0$, $\beta = 1/3$, $\alpha = 0$ when no actions have been taken and strategy $\gamma = 1$, $\beta = 2/3$, $\alpha = 1/3$ when the first player checked and the second player bet. This effectively results in a strategy consisting of $\alpha = 0$, $\beta = 2/3$, $\gamma = 0$, which we note breaks the previous equilibrium parameterization. The best response to this strategy is to never bet with a Jack (and calling with a Queen is irrelevant since the first player never bets). Now, we can calculate the value of the best response to this strategy. When the deal is c_{JQ} or c_{JK}, the first player will check and result in a value of -1. When the deal is c_{KQ} or c_{KJ}, both players check and the first player wins 1. When the deal is c_{QK} the first player checks, the second player bets and the first player calls $2/3$ of the time, resulting in a value of $-2 * 2/3 - 1 * 1/3 = -5/3$. When the deal is c_{QJ} both players check, resulting in a value of 1. The total value of the game to the first player is then $1/6*(-1-1+1+1-5/3+1) = -2/18$, twice the amount an equilibrium loses. Thus, we see how switching between strategies can result in worse play. Despite the lack of theoretical guarantees, we still believe this concept could be beneficial because the theoretical guarantees were already lost when we chose to perform abstraction.

5.4 Cover Set

When playing no-limit poker, understanding the pot size and bet amount is very important. As seen in Chapter 4, a poor understanding of these concepts leads to exploitable agents. Thus, we wish to create an agent that does not have this drawback, but it is difficult to do so in a feasible manner. Strategy switching provides a way to better understand a state space by switching between several different agents that specialize in different areas of the game. By creating a suite of agents that together perfectly understand the *important* aspects of the game, it may be possible to create more robust players.

When playing no-limit poker, possibly the two most important things to understand are the bet ratio (the ratio of the bet to the pot size) and the stack ratio (the ratio of the starting stack size to the current pot size). These two ratios allow an agent to understand the pot odds of calling a bet as well as how many more chips it has to bet with. We want an agent that, in all situations in the full game, has a perfect understanding of the bet and stack ratios. This can be done by creating a suite of agents, each designed to handle different bet and stack ratio occurrences. If this suite of agents handles every possible situation in the full game, then we say that the set of agents covers the pot/stack ratio space.

We experimented with creating such a set of agents in the 12-stack Leduc Hold'em game. Each agent used a betting abstraction of type *fcba* where b represents some pot-fraction bet. Because we were dealing with agents using different pot-fractions, all bets (aside from all-in) in the full game were translated to b. Note that this type of translation is different from both abstract and real translation. In order to create the suite of agents, we walked the full game tree to find all the possible pot and stack ratio situations. At every information set we first calculated the bet ratio needed. If

the last action was a bet, then b would be set to the pot-fraction of that last bet. Otherwise, we considered abstractions for all the possible pot-fraction bets the player could make. We then found an abstract game that, given the ante amount, stack size and betting sequence, would result in the correct stack ratio once translation was performed. Each abstract game is uniquely defined by the stack/ante ratio and pot-fraction bet. Thus, each abstract game handled many situations in the full game and it was easy to check if we already had an abstraction that handled the situation we were examining. This resulted in a suite of 172 abstracts games and resulting abstract game strategies, listed in Table A.1.

In order to use these agents, a strategy switching player was created. This player calculates the current bet and stack ratios and checks which agents in its suite can match those ratios after translation has been performed. When not faced with a bet, any bet ratio is allowed. If multiple agents fit the situation, then the player samples uniformly from those agents. Additionally, we experimented with using subsets of the full cover. In this situation each agent was weighted according to how closely it matched the given bet and stack ratios, and the strategy switching player samples from these agents according to these weights. Specifically, given ratios a, b with $a < b$, the weight associated with how close those ratios are is a/b. If the last action was not a bet, then the weight is the ratio of the stack ratios, otherwise it is the product of the ratios for the stack and bet ratios.

5.5 Results

Table 5.1 shows the exploitability of a variety of cover sets in the 12-stack Leduc Hold'em game as well as the exploitability of the $fcpa$ player using both soft and hard translation. Full Cover refers to the full cover of 172 agents. Sub Cover refers to using a handpicked subset of 15 agents, listed in Table A.2. Sample X refers to one of 10 sample sets that were created. Each of these sample sets contains the $fcpa$ player as well as 14 other randomly chosen (without replacement) agents from the set of 172.

Table 5.1 shows the exploitability of various cover players, where $P1$ and $P2$ refer to the exploitability when the knowledgeable opponent is sitting in seat 1 or 2, respectively. we observe that the full cover did not perform the best. It is outperformed by a player using the $fcpa$ abstraction with abstract and soft translation, as well as the hand-picked sub cover and even random sample 8. This means that it can be disadvantageous to add additional strategies to the suite, even if those strategies could provide a better understanding of the full game in certain situations. Additionally, none of these agents came anywhere close to performing near equilibrium, which is unfortunate considering how many games had to be solved to create these players. Overall, the hand-picked sub cover performed the best at 0.64 mb/h exploitable. This suggests that intelligently picking what strategies should belong in the suite used for switching could create a more robust player.

For the next experiment we took two abstractions of the same size in the 200BB Texas Hold'em game and attempted to switch between them intelligently. One of the abstractions (2) has the half-pot

	P1	P2	Average
Equilibrium	0.07	-0.07	0.00
fcpa (real,hard)	1.44	1.04	1.24
fcpa (real,soft)	0.92	0.93	0.93
fcpa (abstract,hard)	1.14	0.62	0.88
fcpa (abstract,soft)	0.90	0.59	0.75
Full Cover	0.62	0.93	0.78
Sub Cover	0.73	0.55	0.64
Sample 1	1.10	0.64	0.87
Sample 2	0.99	0.56	0.78
Sample 3	1.75	0.89	1.32
Sample 4	1.94	1.04	1.49
Sample 5	0.82	0.76	0.79
Sample 6	1.83	0.97	1.40
Sample 7	1.91	1.16	1.54
Sample 8	0.69	0.67	0.68
Sample 9	1.96	0.92	1.44
Sample 10	2.12	1.47	1.80

Table 5.1: Exploitability of various cover sets in Leduc Hold'em in sb/h

bet option and a smaller card abstraction than the other (1). The concept is that until a half-pot bet is observed in the game, the player would rather use a strategy with a larger card abstraction. This led to the development of two coach players. The Defensive Coach plays strategy (1) until a half-pot bet is observed from the opponent, at which point it switches to strategy (2). The Full Coach plays the same as the Defensive Coach except that it also switches to strategy (2) if that strategy suggests making a half-pot bet with some probability (> 0.001). Ideally these players would perform better than both strategy (1) and (2).

	(1)	(2)
Full Coach	-348	-32
Defensive Coach	0	-125
(1) fcpea.ir169.pub20x75hs2x36hs.pub3x50hs2x18hs.900	0	-36
(2) fchpea.ir169.pub20x36hs2x5hs.pub3x20hs2x3hs.60	36	0

Table 5.2: Switching results for half pot bets in the 200BB stack game in mb/h

Table 5.2 shows the results of the coach players. It appears that the more often the coach players switch between the two strategies, the worse they perform. The Defensive Coach ties strategy (1) since it never actually switches to strategy (2). However, it loses by a significant amount (125 mb/h) to strategy (2), roughly 3.5 times as much as strategy (1) loses to strategy (2) (36 mb/h). The Full Coach performs much better against strategy (2), though it still loses by 32 mb/h. This performance is mainly due to the fact that it plays strategy (2) most of the time and only occasionally uses strategy (1). Conversely, it performs extremely poorly against strategy (1), as it switches between the strategies more frequently since its opponent never makes a half-pot bet.

5.6 Conclusion

Overall, the concept of switching between strategies within a hand produced mixed results. On one hand, we observed that using a coach to switch between two strategies according to the public information performs worse than either strategy independently. This reinforces the theoretical results that suggest that switching strategies mid-game produces non-equilibrium play. On the other hand, we were able to produce more robust players when switching between strategies based upon different betting abstractions in the Leduc game. Although the resulting players are not as close to equilibrium as originally hoped, they are still more robust than any player generated using prior methods. This is a small consolation, however, as these cover players require the generation of many strategies whereas soft translation requires only one.

It is possible that further developing the type of translation used by the cover players and choosing a better suite of strategies could improve the robustness of the final player. However, the scalability of this method discourages us from further investigating what will most likely be additional minor gains in the Leduc game. In the full Texas Hold'em game, it is not feasible to create so many strategies or even strategies with certain bet sizes. The concept could still be implemented, but it would be restricted to a small number of carefully chosen abstractions and a significant amount of time and resources would be needed to create these players. In the mean time, our CPU cycles are better spent elsewhere.

Chapter 6

Conclusion

In this dissertation we examined the concept of state translation and how it is applied in no-limit poker. We defined functions that can be used to implement translation and described the properties, such as maintaining internal consistency, that we want these functions to have. We then described the current translation concept being used, hard translation, and developed a new method, soft translation, that produces more robust players. Having a better understanding of how translation works allows us to produce better players and design better translation systems in the future.

Along with formally describing translation, we examined how the no-limit Polaris agents worked. Due to the nature of the no-limit game, we found that using card abstractions with more immediate information provided a direct benefit to the player. We also found that, despite their cost, adding additional betting options significantly improved the play of the agents. When analyzing the translation used by Polaris, we found many strange behaviors that, after being fixed, resulted in better agents. This analysis also exhibited some of the strengths translation provides. The first strength is the fact that a player's behavior can be changed without modifying the underlying strategy used. This meant that we could modify the translation function to produce better play without solving the abstract game again. The second strength is the ability of the player to take actions in the real game that are not legal in the abstract game. We saw how using the concept of abstract translation, where we interpret the real situation in terms of the abstract game, performed better than real translation, where we interpret the real situation as given. It seems that keeping the real state as close to an abstract state as possible allows the agents to better understand the situation.

Finally, we investigated switching between strategies within a hand. The concept is that if the abstraction underlying each strategy is different, then each strategy understands a different portion of the game space and we can switch to the best strategy for the given situation. Although this technique has the potential to produce the most robust players, the results for this concept were not as good as hoped considering the scalability issues it has.

6.1 AAAI No-Limit Competition Results

6.1.1 2008 Competition

In the 2008 competition we submitted an agent using the ir169.72hs2x2hs.12.12 card abstraction and the $fcpta$ betting abstraction. This agent also had all of the translation fixes as detailed in Section 3.3.2 but used hard translation. Table 6.1 shows the results of the 2008 no-limit competition. Our agent (Hyperborean08) won the competition, beating all of the other opponents. Most importantly, it beat BluffBot 3.0 by 109 mb/h, whose predecessor we lost to in 2007 (BluffBot 2.0). In 2007 we had submitted an agent using the $fcpa$ betting abstraction and the 8.8.8.8 card abstraction. Table 3.3 contains both our 2008 agent (strategy (2)) as well as an agent very similar to our 2007 agent (strategy (6), which differs only in an extra ten-pot bet). The 2008 agent beat the other agent by 193 mb/h. In the 2007 competition (Table 1.1), BluffBot 2.0 beat our agent by 237[1] mb/h. Since the 2008 agent beat the 2007 agent by less than BluffBot beat the 2007 agent and we assume that BluffBot 3.0 was better than BluffBot 2.0, we claim that the translation fixes were the determining factor in defeating BluffBot 3.0 (as we also claim that the same fixes would have resulted in our 2007 agent defeating BluffBot 2.0 had they been implemented at that time).

	(1)	(2)	(3)	(4)	Avg
(1) Hyperborean08		0.109	0.625	2.13	0.95
(2) BluffBot 3.0	-0.109		0.611	2.566	1.02
(3) Tartanian2-Beta	-0.625	-0.611		5.371	1.38
(4) Ballarat	-2.13	-2.566	-5.371		-3.36

Table 6.1: Performance results of the no-limit aspect of the 2008 Computer Poker Competition in sb/h

6.1.2 2009 Competition

In the 2009 competition we submitted two agents since there was now both an equilibrium competition, in which the players attempt to beat all their opponents, and a bankroll competition, in which the players attempt to make as much money as possible. These agents used the strategies (1) and (3) from Table 3.2. It is also important to note that the 2009 competition was run in the 200BB game, whereas the 2007 and 2008 competitions were run in the 500BB game. Our equilibrium submission was an agent using the fchqpwea betting abstraction and the ir169.ir72hs2x3hs.ir72hs3x3hs.216 card abstraction and employed soft translation.

Our bankroll submission was an exploitative agent that attempted to use many of the exploitative techniques that were described in this dissertation. First, the agent would perform exploration by raising different amounts and observing the opponent's response. After an initial period, the agent would then attempt to exploit the model it had built of its opponent. Specifically, the agent would

[1] Due to how close the top three competitors were, additional matches were run for only those agents. After these matches, BluffBot was beating Hyperborean07 by 380 mb/h

analyze whether the model would be exploitable using methods similar to the ones employed by the naïvePA and +- players. If it believed such techniques would work, it would employ the method it predicted would produce the highest value. Otherwise, it reverted to an underlying strategy using soft translation. This strategy used the fchpea betting abstraction and the ir169.dir-pub20x36hs2x5hs-20x1.dir-pub3x20hs2x3hs-3x1.60 card abstraction.

	(1)	(2)	(3)	(4)	(5)	Avg
(1) Hyperborean-BR		-0.06	0.09	2.12	3.53	1.42
(2) Hyperborean-Eqm	0.06		0.09	0.47	0.46	0.27
(3) BluffBot4	-0.09	-0.09		0.19	0.22	0.06
(4) Tartanian3RM	-2.12	-0.47	-0.19		-0.11	-0.72
(5) Tartanian3	-3.53	-0.46	-0.22	0.11		-1.03

Table 6.2: Performance results of the no-limit aspect of the 2009 Computer Poker Competition in sb/h

Table 6.2 shows the results of the 2009 competition. Our agents (Hyperborean) took first and second in both the bankroll and equilibrium competitions. We see that the equilibrium agent (Eqm) defeated all other opponents, successfully defending against the exploitative methods employed by the bankroll agent (BR). We also observe that the bankroll agent managed to win roughly 5 times as many chips on average as the equilibrium agent. This was largely due to the fact that it defeated the Tartanian agents by 2.12 and 3.53 sb/h. Once again, a player that folds every hand loses at most by 0.75 sb/h. This shows that even state of the art agents are susceptible to exploitative techniques.

6.2 Future Work

Despite the improvements based on the concepts in this dissertation, there is still more work to be done. It is possible that further development of many of the concepts discussed could result in even more robust agents. However, there are several other items that could greatly improve a no-limit agent that were not discussed.

6.2.1 No-Limit Man-Machine Match

At some point we wish to play our no-limit agents against professional players like we did with our limit agents [5]. However, our resident poker expert believes that our current agents are not strong enough. This seems strange, as the card abstraction being used by the no-limit agents is a finer abstraction than the one used by the limit agents in the last limit man-machine competition. Thus, it seems that either the card abstraction being used is still not good enough or the betting abstraction being used is still too small.

We showed in Section 3.2 how a no-limit agent benefited from using a different abstraction than a limit agent. However, we are still unsure of what type of abstraction would be best. Ideally we would like a method for evaluating abstractions that does not require us to explicitly compute

strategies within the abstractions to test them. Section 3.2 also showed us that having many betting options can greatly increase the performance of an agent. Looking at the $fchqpwea$ abstraction, we see that despite having a card abstraction several hundreds of times smaller, it plays as well as the agent using the $fcpea$ abstraction. Thus, it is possible that the agents still do not understand the betting space enough to be competitive with human players.

Regardless of what we have seen so far, it may simply be the fact that we have not extracted the important details out of the no-limit game yet. We create abstractions based upon educated guesses, but perhaps these guesses are simply wrong. It is also possible that the abstractions being used are too small to be competitive with human players, and that we simply need to use larger abstractions (which becomes a problem of algorithmic complexity and computing resources). In addition, we also have the problem that it will be much more difficult to obtain statistical significance in such a match. Looking at Table 2.3, the lowest standard deviation we achieved in no-limit was roughly 10 sb/h. This amount is around 5 times as much as the 1.93 sb/h achieved by DIVAT in limit, meaning that we would need at least 25 times as many hands as we did when performing the limit competitions. No matter what the cause, more work still needs to be done before a champion level no-limit player is possible.

6.2.2 Imperfect Recall Betting

Imperfect recall allows one to forget previous information, thus allowing more information to be known at any point in time. When creating card abstractions, this technique was used to increase the number of buckets on each round, since the bucket would be forgotten at the next round. This produced significant improvements for no-limit agents (see Section 3.2). The same concept can be applied to the betting sequence, except that it is more difficult to implement.

The concept of imperfect recall betting is that we forget the specifics of the betting sequence but remember the important properties. For instance, we could only remember the size of the current pot and the bet faced (if any). This would allow the player to have many different betting options without significantly increasing the size of the state space, since many different betting sequences would map to the same {pot,bet} information set. There are many ways this could be done, and its possible that it could provide improvements as drastic or better than imperfect recall on the cards did.

The problem with imperfect recall betting is that it is difficult to implement. Although the state space is much smaller, walking the entire game tree (now a directed acyclic graph) would still require as much work as the perfect recall version of the abstraction. With cards, one can get around this by sampling hands, which is not that difficult since the dealing is controlled by chance and is independent of any player actions. To do the same with player actions, we end up repeatedly playing hands of poker instead of walking the game tree. A new version of CFR, dubbed MCCFR [16], allows one to use imperfect recall on action sequences. However, this has not been applied to

poker yet. No-limit poker seems like a prime testbed for this concept.

6.2.3 Translation as Transfer Learning

In Chapter 4, translation is defined as a method for using a strategy from an abstract game to play in the unabstracted game. However, there is no reason that these two games have to have this relation. The definitions and properties given for translation work equally well for any two games. We could say that we wish to play in game Γ using a strategy for game Γ', with no assumption of any relation between Γ and Γ'. A translation function could certainly be defined, but the question is how to define it such that it results in a good player.

Given two games Γ and Γ', it may be possible to analyze their game spaces to find a suitable translation function between the games. Among two similar games, for instance two different types of poker, it would likely not be very difficult to define such a function. However, it may be the situation where we want to use one strategy for many different games, for instance in the General Game Playing Competition [28]. Is it possible to design a system that would automatically define a good translation function between some base game and the game being played?

Instead of computing a solution for one game to be used in many other games, it may be wise to compute solutions to several different games. One could then analyze the game being played and use the strategy that plays in the most similar game. This concept is similar to the strategy switching concept described in Chapter 5, except that here we would not be switching strategies within a game, but rather between games. Thus, we do not lose any of the theoretical guarantees equilibrium computations allow us. As we did in Chapter 5, we would need to define the set of games to compute strategies for and to use in play, which is another problem in itself.

Bibliography

[1] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 661–668, 2003.

[2] Darse Billings and Morgan Kan. A tool for the direct assessment of poker decisions. *The International Association of Computer Games Journal*, 29(3):119–142, 2006.

[3] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Poker as an experimental testbed for artificial intelligence research. In *Canadian Society for Computational Studies in Intelligence (AI)*, pages 228–238, 1998.

[4] Michael Bowling, Michael Johanson, Neil Burch, and Duane Szafron. Strategy evaluation in extensive games with importance sampling. In *Proceedings of the 25th Annual International Conference on Machine Learning (ICML)*, pages 72–79, 2008.

[5] University of Alberta Computer Poker Research Group. The second man-machine poker competition. http://www.manmachinepoker.com.

[6] George B. Dantzig. *Linear programming and extensions*. Princeton University Press, 1963.

[7] Andrew Gilpin, Samid Hoda, Javier Peña, and Tuomas Sandholm. Gradient-based algorithms for finding nash equilibria in extensive form games. In *3rd International Workshop on Internet and Network Economics (WINE)*, pages 57–69, 2007.

[8] Andrew Gilpin and Tuomas Sandholm. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 50–57. AAAI Press, 2007.

[9] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sorensen. A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems (AAMAS)*, pages 911–918, 2008.

[10] John A. Hartigan and Megan A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[11] Feng hsiung Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002.

[12] ILOG Inc. *CPLEX 9.0 User's Manual*, 2003.

[13] Michael Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, University of Alberta, 2007.

[14] Morgan Kan. Postgame analysis of poker decisions. Master's thesis, University of Alberta, 2007.

[15] Harold W. Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

[16] Marc Lanctot, Kevin Waugh, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. *Conference on Learning Theory (COLT) Workshop on Online Learning with Limited Feedback*, 2009.

[17] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[18] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[19] John A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, 1965.

[20] Yu Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM J. on Optimization*, 16(1):235–249, 2005.

[21] John Von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[22] Martin J. Osborne. *An introduction to game theory*. Oxford University Press, 2002.

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

[24] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd ed.)*. Pearson Education Inc., 2003.

[25] Teppo Salonen. Bluffbot - poker bot world champion. http://www.bluffbot.com/.

[26] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. Chinook: The world man-machine checkers champion. *AI Magazine*, 17(1):21–29, 1996.

[27] David Schnizlein, Michael Bowling, and Duane Szafron. Probabilistic state translation in extensive games with large action sets. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 278–284, 2009.

[28] Stanford University. The 2008 AAAI general game playing competition. http://games.stanford.edu/competition/competition.html.

[29] Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220 – 246, 1996.

[30] Kevin Waugh, David Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathology in extensive games. In *Proceedings of the 8th international joint conference on Autonomous agents and multiagent systems (AAMAS)*, pages 781–788, 2009.

[31] Martin Zinkevich, Michael Bowling, Nolan Bard, Morgan Kan, and Darse Billings. Optimal unbiased estimators for evaluating agent performance. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, pages 573–578. AAAI Press, 2006.

[32] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1729–1736, 2008.

[33] Martin Zinkevich and Michael Littman. The AAAI computer poker competition. *Journal of the International Computer Games Association*, 29, 2006. News item.

Appendix A

Switching Data

Stack/Ante Ratio	Pot Fraction Bet
12.000000	0.500000
12.000000	1.000000
12.000000	1.500000
12.000000	2.000000
12.000000	2.500000
12.000000	3.000000
12.000000	3.500000
12.000000	4.000000
12.000000	4.500000
12.000000	5.000000
9.000000	0.250000
7.080000	0.166667
5.821782	0.125000
4.941176	0.100000
4.269231	0.083333
3.756522	0.071429
3.355932	0.062500
3.027027	0.055556
2.769231	0.050000
6.534653	0.111111
7.080000	0.125000
13.747573	0.187500
7.603960	0.142857
5.346535	0.111111
14.422018	0.214286
25.584906	0.285714
8.330097	0.166667
5.666667	0.125000
9.980198	0.187500
15.089109	0.250000
25.603960	0.333333
41.294118	0.416667
9.148515	0.200000
5.945946	0.142857
4.352941	0.111111
10.099010	0.214286

16.320000	0.285714
15.647059	0.300000
7.309091	0.187500
25.058824	0.400000
38.400000	0.500000
56.160000	0.600000
10.077670	0.250000
6.294118	0.166667
4.543689	0.125000
10.059406	0.250000
15.339623	0.333333
22.514286	0.416667
16.000000	0.375000
7.111111	0.214286
10.368932	0.285714
24.000000	0.500000
34.111111	0.625000
46.860000	0.750000
62.138614	0.875000
11.040000	0.333333
6.534653	0.200000
4.631579	0.142857
3.600000	0.111111
9.777778	0.300000
5.322581	0.187500
13.900990	0.400000
19.200000	0.500000
25.431193	0.600000
16.000000	0.500000
6.718447	0.250000
9.235294	0.333333
12.233010	0.416667
21.720000	0.666667
6.631579	0.285714
28.396040	0.833333
36.000000	1.000000
44.400000	1.166667
53.702970	1.333333
6.712871	0.250000
4.693069	0.166667
3.634615	0.125000
9.148515	0.375000
4.961538	0.214286
15.140187	0.625000
18.712871	0.750000
22.680000	0.875000
15.000000	0.750000
6.117647	0.300000
3.882353	0.187500
7.722772	0.400000
9.600000	0.500000

11.559633	0.600000
18.000000	1.000000
5.520000	0.333333
6.705882	0.416667
21.000000	1.250000
24.000000	1.500000
27.000000	1.750000
30.000000	2.000000
33.000000	2.250000
6.653465	0.333333
4.685714	0.200000
3.600000	0.142857
2.945455	0.111111
8.000000	0.500000
4.475248	0.250000
9.320388	0.666667
4.228571	0.285714
10.641509	0.833333
13.320000	1.166667
14.640000	1.333333
5.227723	0.375000
3.495146	0.214286
6.000000	0.500000
6.742857	0.625000
7.500000	0.750000
8.235294	0.875000
4.320000	0.400000
4.800000	0.500000
5.280000	0.600000
3.657143	0.416667
2.514286	0.050000
2.731707	0.055556
2.285714	0.050000
2.970297	0.062500
2.470588	0.055556
2.086957	0.050000
3.314286	0.071429
2.654867	0.062500
2.244604	0.055556
1.920000	0.050000
3.657143	0.083333
2.888889	0.071429
2.379310	0.062500
2.000000	0.055556
1.730769	0.050000
4.117647	0.100000
3.140187	0.083333
2.532110	0.071429
2.117647	0.062500
2.910891	0.125000
1.811321	0.055556

2.423077	0.111111
1.570093	0.050000
4.680000	0.125000
3.445545	0.100000
2.705882	0.083333
3.540000	0.166667
2.213592	0.071429
2.823529	0.142857
1.882353	0.062500
2.340000	0.125000
2.823529	0.187500
1.631068	0.055556
1.981651	0.111111
1.431193	0.050000
5.307692	0.166667
3.728155	0.125000
4.500000	0.250000
2.880000	0.100000
3.360000	0.200000
3.840000	0.300000
2.330097	0.083333
2.653846	0.166667
3.000000	0.250000
3.326733	0.333333
1.945946	0.071429
2.192308	0.142857
2.446602	0.214286
2.679612	0.285714
1.680000	0.062500
1.864078	0.125000
2.057143	0.187500
1.471698	0.055556
1.621622	0.111111
1.320000	0.050000

Table A.1: Full cover set for 12-stack Leduc Hold'em

Stack/Ante Ratio	Pot Fraction Bet
12.000000	0.500000
12.000000	1.000000
12.000000	1.500000
12.000000	2.500000
12.000000	4.000000
5.520000	0.333333
1.920000	0.050000
2.340000	0.125000
10.059406	0.250000
15.089109	0.250000
24.000000	0.500000
18.000000	1.000000
7.080000	0.166667
6.000000	0.500000
2.679612	0.285714

Table A.2: Sub cover set for 12-stack Leduc Hold'em