Data: You had nothing!
Geordi: He bluffed you, Data.
Data: It makes very little sense to bet when you cannot win.
Riker: But I did win, I was betting that you wouldn't call.

– Star Trek: The Next Generation, Episode "The Measure of a Man"

University of Alberta

AUTOMATED ABSTRACTION OF LARGE ACTION SPACES IN IMPERFECT INFORMATION EXTENSIVE-FORM GAMES

by

John Hawkin

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

To my wife, Jenn.

Abstract

An agent in an adversarial, imperfect information environment must sometimes decide whether or not to take an action and, if they take the action, must choose a parameter value associated with that action. Examples include choosing to buy or sell some amount of resources or choosing whether or not to move combined with a distance for that movement. This problem can be expanded to allow for mixing between multiple actions each with distinct associated parameter values and can be expressed as an imperfect information extensive form game. This dissertation describes a new automated method of abstracting the action space of such decision problems. It presents new algorithms for implementing the method, provides some theory about how the method relates to Nash equilibria in a small poker game and assesses the method using several poker games. One of these algorithms was used in the creation of an agent that won one of the divisions of the 2012 Annual Computer Poker Competition. An improvement upon this algorithm produced an action abstraction of two-player no-limit Texas hold'em poker that out-performs a state-of-the-art action abstraction while requiring less than 40% of the memory. The resulting agent had the best overall results in a round robin tournament of six top two-player no-limit Texas hold'em agents.

Acknowledgements

There are many people I would like to thank for helping me during my PhD. My supervisors, Rob and Duane, were always extremely patient, kind, and generous with their time. There were very understanding during the many times I had to delay my work for one reason or the other. I am very lucky to have worked under the tutelage of such skilled scientists.

The many members of the Computer Poker Research Group in my department were all extremely helpful, and much of this work could not have been done without them. They deserve a ton of credit, not just for guidance, but for all of the work they did that I was able to build upon. It was a joy to work with such brilliant people. They were also great colleagues for the many years I spent in the department. Thank you to each of you.

Thank you to my family, who saw me through the good and the bad, and put up with my answer of "about another year" once a year, every year, since about 2010, when I was asked how much longer I would take to finish. I am happy to be home and see them much more now.

To all my other friends in Edmonton – thanks for the memories. I enjoyed this period of my life because I had such great people to hang out with.

Finally, I would like to thank my wife Jenn. She is always there when I need her. She is always understanding. A better person I have not met. Without her, I'm not sure I would have been able to do this. Jenn, I love you.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Games are a fantastic test bed for Artificial Intelligence (AI) research. They provide an excellent domain in which to work, with clearly defined actions and goals. If we can learn to play different types of games at a high level, then we can learn to behave intelligently in real world domains that can be modelled as games. Some examples of such real world problems include scheduling port security rotations, choosing investment strategies and ordering troop movements in military conflicts. Games can vary in difficulty from very trivial games such as tic-tac-toe to very complex games such as chess, poker and go. This spectrum of difficulty allows researchers to develop techniques, test them on small games, and eventually apply them to larger games. Games that are played competitively among humans provide another appealing option — testing of agents created using AI techniques against human players, from amateurs to world class professionals.

The games that have received the most attention from the AI community in the past have been games of perfect information such as chess and checkers. In these games the entire state of the game is known by both players. Games of imperfect information such as poker, on the other hand, present a different type of challenge for computer scientists — the opponent's cards are unknown, they do not know our cards, and neither party knows what cards will be dealt in future rounds. These properties, namely imperfect information and stochasticity, are common in real world situations. For example, a robot navigating foreign terrain is only aware of what it can detect with its sensors (imperfect information), and must be able to adapt to random fluctuations of the environment (stochasticity).

Two player limit Texas hold'em has been the subject of a great deal of recent work in AI [6, 37, 21, 24, 68, 65]. Though there are other popular poker games such as Omaha and Stud, Texas hold'em is by far the most popular poker game today, partly due to its rich strategic complexity [13]. This popularity and complexity make Texas hold'em a good choice of poker variants as a testbed for AI research. A great deal of progress has been made in recent years creating poker agents (known as "bots") that play two player limit Texas hold'em. In the summer of 2008 a man machine match was held with some of the top two player limit Texas hold'em players in the world, resulting in a

victory for the computer [2]. While many of the AI techniques used to develop two player limit Texas hold'em bots can be applied to two player no-limit Texas hold'em, the latter game includes an additional challenge — it is much larger. Heads up no-limit Texas hold'em with a stack size of 200 big blinds and blinds of 50 and 100 chips (the variant played in the computer poker competition [32]) has $\sim 10^{164}$ game states [38], while in comparison two player limit Texas hold'em has $\sim 10^{18}$ game states [6].

Interesting games such as Texas hold'em are too large to be solved directly, so first we must apply abstraction to create a smaller game. There are two main types of abstraction — action abstraction and state abstraction (throughout this dissertation we call state abstraction of poker games "card" abstraction). When navigating a robot, a simple action abstraction would be something like move backward, move forward, turn left and turn right. State abstraction might involve classifying surfaces as either "smooth" or "rough" and treating all surfaces of one type the same way. In a poker game, action abstraction involves limiting the actions that we allow either ourselves or the opponent to make. For example, it is common in no-limit poker games to decide on a small number of bet sizes that we will allow ourselves to make at all points in the game. This means that we only have to consider what happens when we make one of these bets. An example of state (card) abstraction in poker would be treating similar cards, say a pair of Aces and a pair of Kings, the same way everywhere in the game.

State abstraction of large poker games is essential in any attempt to solve them. In addition to this, the first equilibrium-based two player limit hold'em bots also used action abstraction, reducing the number of bets allowed each in a round from four to three [6]. Modern two player limit hold'em bots no longer take this approach, as better performance can be achieved by fully representing all actions and only abstracting the state space of the game [6]. The explosion in the size of the no-limit game is caused by the large number of actions each player may choose from at every point in the game. One approach to solving the no-limit game is to abstract the action space of the game in the way I described above, removing the majority of the actions until the game is similar in size to limit Texas hold'em, and then applying the same solution techniques that have been so effective in the limit game. This is the approach I use in this dissertation — I present an algorithm that takes a game with a large continuous or near-continuous set of actions and intelligently picks a small subset of these actions to keep, creating a much smaller abstraction of the full game. In this dissertation I focus on the poker literature that relates to action abstraction in agents that play approximate equilibrium strategies. For a good general review of literature related to computer poker see Rubin and Watson [56].

1.2 Poker literature on no-limit bet sizing

There is a great deal of poker literature that deals with the issue of bet sizing in no-limit Texas hold'em. The books No-limit hold'em: Theory and Practice [62], Heads-up No-limit Hold'em [51],

Professional No-limit Hold'em: Volume I [13], Harrington on Hold'em: Volume I and II [27, 28] and Harrington on Cash Games Volume I and II [29, 30] all emphasize the value of picking different bet sizes in different situations. While some offer general rules of thumb with regard to bet sizes [13, 27, 28, 29, 30], others emphasize the situations in which bet sizes should be adjusted up or down depending on the current game state [51, 62].

In Harrington on Hold'em: Volume II [28], Harrington advocates a jam-fold strategy be used when the stack to big blind ratio hits 7.5, and sometimes when the ratio is between 7.5 and 15. Similarly in the Mathematics of Poker by Chen and Ankenman [9] the authors suggest that using a jam-fold strategy is close to optimal with stacks of about 11 or 12 big blinds for the small blind and up to 13 or 14 for the big blind. Chen and Ankenman calculate near optimal jam-fold strategies for stacks of up to 50 big blinds, and their results are in good agreement with the work of Miltersen and Sorensen mentioned above in Section 3.8.1 (which was for 6.67 big blinds). Additionally, Chen and Ankenman include a bound on their suboptimality [9].

Chen and Ankenman also devote a couple of chapters of The Mathematics of Poker to the study of no-limit bet sizing in Kuhn poker [9]. They derive analytical solutions for three no-limit Kuhn games - half street Kuhn, Kuhn with no raises allowed and Kuhn with no restrictions. They show that Nash equilibria in these games use a variety of bet sizes, and they go on to show that in certain simple multi-street games a bet size of pot is optimal. It should be noted that the derivation of the analytical solution to no-limit Kuhn poker is seven pages long, and their approach does not scale well and thus cannot be directly applied to two player no-limit Texas hold'em. I use their analytical solutions to test our algorithm for generating action abstractions, as I will show in Section 4.4.

Chris Ferguson, a world series of poker main event champion, is well renowned in the poker community for his bet sizing abilities. In the book The Full Tilt Poker Strategy Guide [10], he wrote a section entitled "How to bet" in which he describes his bet sizing methodology. He advocates betting different amounts in the pre-flop based on a player's position only (in a nine or ten handed game). Thus for two player games his suggestion is to use only one bet size, independent of your private cards. On further rounds he bases his bet size on the texture of the board, what he believes his opponent's possible hands could be, and his best guess as to what his opponent believes he may have. He then determines an appropriate bet size, all the while ignoring his private cards. He only uses his private cards to determine *if* he will bet. An advantage of this approach is that it incorporates information hiding — it is impossible to deduce private cards from bet sizes if the two are completely independent.

1.3 Problem significance and difficulty

An abstraction of the legal actions in two player no-limit Texas hold'em leads to problems when such a strategy is used to play the full game, as actions made by the opponent that are not within this abstraction must be handled. This is known as the translation problem, and it has received some

attention in recent years [58, 59]. While solving the translation problem deals with interpreting bets made by the opponent, the question of what bet sizes a player should make for themselves has been given little attention. Many modern no-limit poker bots use simple betting schemes, such as only ever betting the the number of chips that are currently in the pot (called a pot bet) or all of their chips (known as an all-in bet) [25, 58]. A great deal of value can be gained from choosing appropriate bet sizes - in the small poker game of no-limit half street Kuhn poker, for example, the betting player has the advantage if they choose their bet size well. If, however, the betting player chooses a bet size of pot, their advantage is reduced to zero (see Section 4.2.1). This is assuming that both players play according to a Nash equilibrium strategy for the given bet sizes — that is, they both choose a strategy such that neither player can deviate from this strategy and gain value. The bettor has lost their advantage by selecting an inferior bet size [9].

Another example of the importance of picking good bet sizes comes from work done by Schnizlein [58]. Schnizlein did experiments in 100 and 200 big blind no-limit Texas hold'em, comparing approximate Nash equilibrium strategies in abstractions of these games that allowed only a few hand-picked bet sizes. In the 100 big blind game his experiments showed that adding the option of making a half pot bet once per round can lead to a significant improvement in performance over bots using abstractions which allow only pot and all-in bets. He then did an experiment in the 200 big blind game keeping the size of the abstract game constant and alternating between allocating more space to the card abstraction and the action abstraction. As with the 100 big blind game allowing a single half pot bet each round in addition to pot and all-in bets led to the best performance, even though this increased the size of the abstraction by 15.7 times, and thus a much smaller card abstraction was used. The most interesting result of this experiment, however, was that an agent which allowed half pot, three quarter pot, pot, 1.5 pot and all-in bets performed roughly the same as an agent allowing only pot and all-in bets but using a much more refined card abstraction [58]. The extra bet sizes increase the size of the game by 248 times, and thus for this experiment the bot using only pot and all-in bets used a card abstraction that was 248 times bigger.

This result suggests that the choice of bet sizes is of great importance in creating an effective no-limit poker bot. An analytical approach to solving for bet sizes in no-limit poker games was successfully applied to the very small game of no-limit Kuhn poker [9], however scaling such methods to a large game such as two player no-limit Texas hold'em is intractable. One must use abstraction, as even if we could solve a game with $\sim 10^{164}$ bet sizes, there would not be enough memory to store a strategy to play the game. To this author's knowledge, there has been no work on automating the action abstraction of no-limit poker other than the work presented in this dissertation.

1.4 Thesis Organization and Contributions

The action space of no-limit Texas hold'em is extremely large. At each decision point the player must decide between a large number of different bet sizes. This can be approximated by instead

having them simply choose whether or not they will bet, and if they do bet, the amount that they wish to bet. This is now part of a more general class of problems in game theory, where an agent in an adversarial, imperfect information environment must decide whether or not to take an action and, if they take the action, must choose a parameter value associated with that action. Examples include choosing to buy or sell some amount of resources or choosing whether or not to move combined with a distance for that movement. This model can be expanded to allow for mixing between multiple actions each with distinct associated parameter values. The contribution described in this dissertation is an automated method of abstracting the action space of such games, with an emphasis on no-limit poker games.

In Chapter 2, I explain the rules and terminology of the no-limit poker games we use as test-beds throughout the dissertation. I begin Chapter 3 by defining relevant background material in the field of game theory and discussing the game theory concepts that are important to this work. I then introduce related work on domains with large action spaces. I finish the chapter with a summary of the history of computer poker research and a description of the current state of the art in this field.

In Chapter 4, I show that, using a new transformation and accompanying regret-minimizing algorithm, I can compute the bet sizes used by Nash equilibrium strategies in small no-limit games, without solving those games directly. I introduce a new transformation that takes a game of the type described above and creates a new multi-agent game consisting of two teams. Each team has a single decision agent and possibly multiple parameter-setting agents, one for each parameter. The transformation is designed such that if we define a strategy for each parameter-setting agent in the transformed game, these map directly to parameter values in the original game. I demonstrate the usefulness of this transformation empirically and analytically in small no-limit poker games. I show that when this transformation is applied to half-street no-limit Kuhn poker, a Nash equilibrium of the resulting game has the useful property that the strategy of the parameter-setting agent can be mapped to the bet size used in a Nash equilibrium solution of the original game. I then introduce a new regret-minimizing algorithm, which I call $BETS$. This algorithm is based on the CFR algorithm [68], designed specifically to be used on games after applying my transformation. I use $BETS$ on the transformed version of various small no-limit games. I show empirically that, in each case, the resulting strategies for the parameter-setting agents can be mapped to the bet sizes used in a Nash equilibrium of the original game.

In Chapter 5, I consider the problems that arise when applying my transformation to games where the depth of the game tree is large. The $BETS$ algorithm, introduced in Chapter 4, defines a range of possible values for each parameter in the game. The parameter-setting agents may select a value from that range. In Chapter 5 I show that careful range choice is essential to creating a good action abstraction. I explain why selecting appropriate ranges is a non-trivial problem. There are practical reasons why large ranges cannot be used, and in general no single small range is appropriate. I demonstrate a new algorithm, RE–$BETS$, which automatically adjusts ranges using multiple

short runs of $BETS$. I apply RE–$BETS$ to no-limit Leduc poker and no-limit Texas hold'em by fixing the bet sizes of one player and using parameter-setting (bet-sizing) agents to choose the bet sizes of the other player. The resulting abstract games are both smaller than common hand-crafted action abstractions and have higher game value for the player whose bets have been chosen by RE–$BETS$. I finish Chapter 5 by describing how RE–$BETS$ was used to create the agent that won the no-limit bankroll instant run-off division of the 2011 Annual Computer Poker Competition (ACPC).

In Chapter 6, I consider another approach to choosing the end points of the ranges used by the parameter-setting agents on each run of $BETS$. RE–$BETS$ uses fixed-width ranges, where the distance between the endpoints of each range remains a constant fraction of the pot size for all parameter-setting agents. I show that setting the range width in a different way, which I call fixed-ratio ranges, has many advantages over fixed-width ranges. Next I consider generalizing RE–$BETS$ to use more than one parameter-setting agent at each decision node. Combining this with fixed-ratio ranges leads to the $REDM$–$BETS$ algorithm. I show results of $REDM$–$BETS$ applied to a perfect recall card abstraction of no-limit Texas hold'em, using a simple action abstraction for the opposing player. Using two parameter-setting agents per node leads to an increase in value, however using three parameter-setting agents does not improve over two. I conclude Chapter 6 by introducing a technique where I allow multiple parameter-setting agents on the first run of $BETS$, after which I pick the most valuable of these actions and remove the less valuable ones. This leaves multiple parameter-setting agents at the same decision point in only a few places, where each is useful. The resulting algorithm, $REFINED$–$BETS$, leads to a further reduction in the size of my abstractions while maintaining the value achieved by $REDM$–$BETS$, again when applied to a perfect recall card abstraction of no-limit Texas hold'em using a simple action abstraction for the opposing player.

In Chapter 7, I apply the $REDM$–$BETS$ and $REFINED$–$BETS$ algorithms to imperfect recall card abstractions of no-limit Texas hold'em, using two different action abstractions for the opposing player. I show that having more than two parameter-setting agents at each decision node does lead to improved performance in imperfect recall games, unlike the perfect recall game of Chapter 6. I also show that increasing the size of the imperfect recall card abstraction does not significantly change the action abstractions generated by my algorithms. This leads me to the idea of creating action abstractions using small card abstractions, then pairing the resulting action abstractions with larger card abstractions when creating the final production agent. I use the $REFINED$–$BETS$ algorithm and this technique to create an agent for no-limit Texas hold'em that performs very well when compared with other state-of-the-art agents.

The research described in this dissertation makes the following contributions:

- I introduce a transformation and accompanying regret-minimizing algorithm ($BETS$) that is shown experimentally to generate high value action abstractions (Chapter 4).

- I introduce an algorithm that iteratively applies $BETS$, known as RE–$BETS$, which resolves some of the issues with scaling the $BETS$ algorithm to large games (Chapter 5).
- I further refine the RE–$BETS$ algorithm to allow for multiple parameter-setting agents at a single decision node. This leads to the $REDM$–$BETS$ algorithm (Chapter 6).
- I introduce a metric that can be used to rank the importance of parameter-setting agents. This is used in my final algorithm, $REFINED$–$BETS$, to prune the game tree (Chapter 6).
- I show that these algorithms scale well – that when applying these algorithms to small imperfect recall poker games, the resulting action abstractions perform well when paired with larger card abstractions (Chapter 7).
- I apply RE–$BETS$ and $REFINED$–$BETS$ to create state-of-the-art no-limit poker agents (Chapter 5, Chapter 7).

Chapter 2

Poker games

Limit and no-limit poker games have different betting rules. In limit poker at every point in the game where a bet is legal there is only one bet size that can be made. In no-limit, however, a player may bet any integer amount of chips between some minimum bet size and all of their chips. In this chapter I will describe the basic rules of all two player no-limit poker games, and I will define in detail a number of games that I will be using later.

2.1 Rules of two player no-limit poker

This subsection presents a set of rules that are common to all of the two player no-limit poker variants used in this dissertation. Each player starts with the same number of chips, known as their stack. These chips are used for betting throughout the game. Before cards are dealt each player is required to put some number of chips in the pot. If the game uses antes, then each player places the same number of chips in the pot. If the games uses blinds, then one player places a set number of chips known as the small blind in the pot and the other player places the big blind, usually twice the size of the small blind. The players are often referred to by the blind they put in the pot - the small blind player and the big blind player. Typically the size of each player's stack is measured in big blinds or antes — a common stack size for no-limit Texas hold'em poker is 100 big blinds.

Next, cards are dealt to each player, at least one of which is always dealt face down and kept private. This adds the element of imperfect information to the game. After each player receives the same number of cards a betting round begins. If no bet has been made so far in a round then the player first to act may choose between one of two actions:

- *bet* - place some number of chips in the pot, forcing the opponent to either match them (call) or surrender the hand (fold).
- *check* - decline the option to bet.

If a bet has been made, a player has three possible responses (note that in some variations some of these may be disallowed at certain times):

- *fold* - give up on the hand, surrendering the pot to the other player.
- *call* - match whatever bet the opponent has made and end the betting round.
- *raise* - match any bet made by the opponent and increase it, requiring the opponent to match this new bet. A raise is best thought of as a call followed immediately by a new bet.

Now that I have introduced the actions I can explain an additional difference between antes and blinds. When using antes the first player faces no bet, so the valid actions are bet and check. When using blinds the small blind acts as a forced bet, and the big blind acts as a forced raise. Thus the player who placed the small blind in the pot is faced with a bet, and may fold, call or raise. Note that a special situation arises if the player in the small blind elects to call, matching the big blind. The player in the big blind is then given the option to either check, ending the betting round, or bet, forcing the small blind to respond.

Poker games are made up of betting rounds, which always follow the dealing of cards. Each player is given a turn to act (in some test games I suppress this rule). If no bet is made, then the betting round ends and the next set of cards are dealt according to the rules of the game. If a bet or raise is made then the other player must respond to this action before continuing to the next round.

When a player elects to bet in a limit poker game, there is only one bet size they are allowed to make, determined by what betting round they are in. In no-limit poker a player may bet any amount of the chips remaining in their stack, with the restriction that they must at least match the last bet that was made in that betting round. For example if the small blind raises, matching the big blind and betting an additional 10 chips, then if the big blind wishes to raise again they must put in at least 20 chips - calling the bet of 10 chips and betting at least an additional 10 chips. The first bet or raise in each round must at least match the size of the big blind. In games that use antes the initial minimum bet size varies depending on the game. The minimum bet restriction is lifted if the player has less than this minimum number of chips left in their stack, in which case if they wish to bet they must put all of their remaining chips in the pot (this is known as an all-in action). I will sometimes ignore the minimum betting rule when working with small games for testing purposes.

Cards dealt before a betting round are either personal cards, which are only seen by the player they were dealt to and may only be used by that player, or community cards, which may be used by either player[1]. If we reach the end of the final round of betting and neither player has folded, the pot is awarded to the player that can form the best poker hand using their own cards and the community cards. I will assume the reader is familiar with the ranking of poker hands (if not see [27]).

Now that I have defined the rules I will introduce several useful no-limit poker games. Some small no-limit games use fewer cards and thus use a modified version of the usual hand rankings, which I will define in each case below.

[1] In the games I will consider the personal cards are always unseen by the other player, though there are poker games where at least one of these is made public information.

2.2 Kuhn poker

Kuhn poker is a small poker game invented by H. W. Kuhn [45]. It is played with three cards in the deck - one Jack, one Queen and one King. The King has the highest value and the Jack the lowest. Kuhn poker is usually played using antes, though blinds may also be used. In all of my experiments I will use antes. After the antes are paid one card is dealt face down to each player, leaving one card unseen by both players. The game's only betting round then begins. In Kuhn poker no raises are allowed — if one of the players bets, the other player may only fold or call. If the betting round ends with neither player folding then the player holding the highest card wins the pot.

Kuhn poker is very useful as a testbed because it is a very small game, and yet it still retains some of the basic strategic elements of larger games such as Texas hold'em poker [35][2]. Kuhn poker can be extended by allowing players to make an unlimited number of raises (until they run out of chips), and I will sometimes use this extension for testing purposes. The size of the bet in the original game is the same size as the ante, however I generalize the game to a no-limit version by allowing the bet size to be anything up to the stack size.

2.2.1 Half street Kuhn poker

Kuhn poker can be further simplified by not allowing the second player to bet. This means if the first player checks the game ends and the player with the highest card wins the antes. If the first player bets then the second player may either fold or call. This variation of Kuhn poker is known as half street Kuhn poker. As with regular Kuhn poker, I generalize this game to a no-limit version by allowing the betting player to make any bet size up to the stack size.

2.3 Leduc poker

Leduc poker uses a deck with three different ranks and two different suits. The Jack, the Queen and the King are used again, with the same relative values. The game can be played with either antes or blinds. Each player receives a card face down and the first betting round begins. After the first betting round concludes if neither player has folded a community card is dealt. Then the second betting round commences. If neither player folds then the winner is the player who can make the best two card hand. This means if either player has a pair (their private card is the same as the community card) they win, and if not then the player with the highest private card wins.

By default there is no limit to the number of raises players can make in Leduc poker. I sometimes decide to impose a limit in order to reduce the size of the game.

[2] Two such strategic elements are bluffing, or betting when you have a poor hand in the hopes that the opponent will fold, and slow-playing, or not betting when you have a very strong hand to induce a bet from your opponent.

2.4 Texas hold'em poker

The final game I will define is Texas hold'em poker. This game uses a full 52 card deck, with 13 ranks and 4 suits. There are four rounds of play known as the preflop, the flop, the turn and the river. Each player is initially dealt two private cards face down and the preflop betting round begins. Next three community cards are dealt face up in the middle of the table — this is known as the flop. After a betting round another community card is dealt (the turn) and the third betting round commences. The final community card is then dealt (the river) and one more round of betting occurs. Once this round is over both players make the best five card poker hand they can using any five of their two private cards and the five community cards, and the best poker hand wins. As usual if either player folds at any point, the game ends and the pot is given to the other player.

2.4.1 Doyle's game

One hand of poker takes very little time to play, and players usually play several hundred or thousand hands in a row. If both players start with N chips and player 1 wins k chips on the first hand, then typically both players will continue playing with player 1 having $N + k$ chips and player 2 having $N - k$ chips. The value of the game at this point is the same as if both players had $N - k$ chips, since one player cannot bet more chips than the other player has. This is true whether both players are playing a cash game format, where either player may choose to quit at any point, or a tournament format, where both players play until one player has all the chips [9].

Instead of playing either of these formats I use another format known as Doyle's game [32]. If each player begins hand number 1 with a stack of size N, then after this hand we will keep track of which player won or lost chips and reset both players stacks to size N. I do this after each hand so that every hand begins with the same stack sizes. There are two reasons to do this. On a strategic level, the larger the stacks the more strategically complex the game is, due to the larger number of information sets. I wish to work with a strategically complex game. On a practical level a strategy designed for a game with stacks of size 200 cannot be applied to a game with stacks of size 20. Thus to design an agent to play a two player tournament, it would be necessary to solve the game for a huge number of different stack sizes and switch between these strategies each hand depending on how many chips the player with the least chips had. While this is certainly possible, partly because the smaller stack games are much quicker to solve, a lot of computational work is required to play what is considered to be a less challenging game [62].

2.5 Bet and raise sizes in no-limit poker

Bet and raise sizes in no-limit poker games are usually expressed in terms of the current size of the pot [62, 61, 51, 13]. For example, if there are 20 chips in the pot, and a player makes a bet of 10 chips, this is called a half pot bet, or a bet of 0.5 pot. If the other player then responds by putting

50 chips in the pot, this is referred to as a pot-sized raise. The first 10 chips match the initial bet, bringing the pot size to 40 chips, and thus the additional 40 chips make up a pot-sized raise. I express bet and raise sizes in this way throughout this dissertation, and I will use the term "bet" to refer to both bets and raises, and "bet size" to refer to both bet and raise sizes.

Chapter 3

Related work

I begin this chapter by defining extensive form games and various game theory terms that I will be using. Next I introduce the most important and influential algorithms for finding ϵ-Nash equilibria in extensive form games. I then discuss abstraction and outline the usual approach to solving extensive form games. After this I provide a review of the existing literature on solving domains with large or continuous action spaces. I move on to discuss the development of poker agents in limit Texas hold'em. I conclude this chapter with a summary of the state of research on two player no-limit Texas hold'em.

3.1 Extensive form games

An extensive form game consists of:

- N players.
- A set H of histories. Each history h in this set consists of a sequence of valid actions for the game (transitions between states in the game made by players and chance).
- A set $Z \subseteq H$ of terminal histories. A terminal history represents the end of the game — no more actions may be made.
- An information partition $\mathbf{I}_i$ of all choice points in the game for each player i consisting of all of the information sets *I* for that player. An information set for a player is a set of histories that the player cannot distinguish from each another.
- A player function $P(h)$, defined for every history h, that determines whose turn it is. This can be $P(h) = i$, meaning player i's turn, or $P(h) = c$, where c is chance.
- A function f_c defining the probability that chance will take each of the available actions when $P = c$.
- A utility function u_i for each player i assigning a real value to each terminal history. This is the payoff for that player when that terminal history is reached.

Extensive form games can be represented by a figure known as a game tree. A game tree is made up of decision nodes, chance nodes, terminal nodes, and actions. The edges in the tree represent actions, the chance and decision nodes represent information sets and the terminal nodes represent terminal histories. At each decision node player $P(h) = i$ chooses an action. A chance node corresponds to a history h such that $P(h) = c$. At a chance node the actions are chosen according to the function $f_c^I(a)$ for that node. At a terminal node payoffs are given to each player i according to the utility function $u_i(h)$.

3.2 Game theory definitions

In order to play an extensive form game each player must have a strategy. A pure strategy defines a single action to be taken at every information set in the game. A mixed strategy is a probabilistic combination of all pure strategies, chosen prior to playing. A behavioral strategy is a probability distribution over actions defined at each information set. Perfect recall is the assumption that a player remembers all their own past actions, as well as any prior observations when making those actions [67]. Any game where this is not true is known as an imperfect recall game. In perfect recall games any mixed strategy can be represented as a behavioral strategy and vice versa [44]. In imperfect recall games this is no longer the case.

The following are textbook definitions [26]. A strategy profile is a set of strategies, one for each player in the game. Given a strategy profile, a best response is a strategy for one player that maximizes that player's expected utility when played against the strategies of the other players in that strategy profile. A Nash equilibrium is a strategy profile such that no player can improve their utility by changing their strategy. This means that if all players are playing according to the same Nash equilibrium strategy profile, then each player is playing a best response to all other players. An ϵ-Nash equilibrium is a strategy profile in which no player can improve by more than ϵ by changing their strategy. A constant sum game is a game in which the utility of all players sums to a constant at every terminal node. If the utility sums to zero we have a zero sum game. A constant sum game can be converted to a zero sum game by subtracting the constant divided by the number of players from the payoffs of all players at all terminal nodes. In a two player zero sum game if each player plays an equilibrium strategy from any equilibrium strategy profile, each player's expected utility is a fixed value, known as the value of the game for that player.

3.3 ϵ-Equilibrium solvers

Most of the state of the art poker agents that exist today are created using an ϵ-equilibrium solver. These produce ϵ-Nash equilibria for the games to which they are applied.[1]

[1] Note that if abstraction is used the result is an ϵ-Nash equilibrium for the abstract game, not necessarily the full game [66].

3.3.1 Linear programming

Normal form representation

Normal form is a description of a game where, instead of the graphical representation used in extensive form, the game is represented by a matrix. Let us define $\mathbf{x}$ as a vector of length n where each x_i is the probability that player one will play pure strategy i and n is the number of pure strategies available to player one. The equivalent vector of length m for player two is $\mathbf{y}$, where player two may choose from m pure strategies. The expected payoff to the first player can be expressed as

$$x^T Ay \tag{3.1}$$

where A is an n x m payoff matrix. Each entry in this matrix A_{ij} is the utility for player one if they play pure strategy i and player two plays pure strategy j. The elements of vector $\mathbf{x}$ must add to 1, since player 1 must choose a pure strategy each time the game is played, and similarly for $\mathbf{y}$.

A Nash equilibrium can be found by solving

$$\max_x \min_y x^T Ay \tag{3.2}$$

subject to the constraints on $\mathbf{x}$ and $\mathbf{y}$ I have described in this section, as well as $x_i \geq 0$ for all i between 1 and n and $y_j \geq 0$ for all j between 1 and m. This equation and these constraints can be transformed to a standard linear programming form and can then be solved using linear programming techniques, such as the interior point method [49] or the simplex method [52].

Sequence form representation

The development of sequence form represented a huge step forward in solving two player zero sum games [43]. In the normal form representation of a game the payoff matrix contains an entry for every possible combination of pure strategies for both players. This stores a great deal of redundant information — consider a game where player one choses from a actions and then player two responds by picking one of b actions, ending the game. Each of player two's pure strategies defines a response to all possible actions player one could make, and thus player two has ab pure strategies and there are ab columns for player two in the normal form payoff matrix. The payoff matrix stores a^2b values, even thought there are only ab possible sequences of actions in this game. Sequence form removes this redundancy by replacing pure strategies with sequences, which specify a player's moves only along a specific path in the game tree [54]. The number of sequences is thus bounded by the number of nodes in the tree. Koller et al. [43] show that this problem can still be formulated as a linear program, and thus solved using any linear programming method.

3.3.2 Counterfactual regret minimization

The counterfactual regret minimization algorithm (CFR) is currently the most successful and commonly used algorithm for finding ϵ-Nash equilibria in two player poker games [32]. The concept of regret refers to how much more value one would have obtained by choosing a utility-maximizing strategy instead of the strategy that was employed [37]. Note that a large regret value for a strategy means that strategy would perform much *better* than our current strategy. When a game is played repeatedly, average overall regret refers to the average of the differences in utility between the strategies chosen each time the game was played, and the strategy that would have maximized utility over all games played [37]. If the average overall regret of a strategy profile is less than ϵ, then this strategy profile is a 2ϵ-Nash equilibrium. The CFR algorithm uses a concept known as immediate counterfactual regret defined at an information set I, which is a player's average regret for their actions at I over multiple games, assuming they played to reach that information set. Zinkevich et al. show that the sum of immediate counterfactual regret over all information sets bounds the average overall regret [68].

CFR is a self play algorithm that works by using Blackwell's algorithm for approachability to minimize the immediate counterfactual regret at each information set [11]. This is done by considering every action at every information set and calculating the utility that would have been gained or lost if that action was taken instead of what was actually played. This value is added up each iteration for each action. Zinkevich et al. call this sum the cumulative regret of each action [68]. For each iteration, the absolute value of these cumulative regrets are added together, and both players' strategies are changed so that they take each action at each information set with probability proportional to that action's contribution to this sum. If an action's cumulative regret is negative it is ignored. For example if the cumulative regrets over all iterations at a given information set are 40 for folding, 60 for calling and -25 for raising, then the strategy at that information set on the next iteration will be $P(f) = 0.4$, $P(c) = 0.6$ and $P(r) = 0$. Zinkevich et al. show that if this algorithm is applied to any perfect recall two player zero sum game, the average strategy profile converges to an ϵ-Nash equilibrium, where ϵ shrinks as the number of iterations increases [68].

The CFR algorithm is often run using sampling of the state space, and can be generalized to use sampling of the action space as well ([18, 40, 46]). When CFR was first applied to poker the authors were able to solve games with $\sim 10^{12}$ game states, two orders of magnitude larger than was previously possible using linear programming techniques on the sequence form representation of the game [68].

3.3.3 Gradient-based approach

Gradient-based algorithms attempt to solve the sequence form representation of an extensive form game (see Section 3.3.1) by starting from an arbitrary point in the strategy space and descending to the saddle point. There is a problem with trying to solve this equation for poker games —

$\min_y x^T Ay$ and $\max_x x^T Ay$ are not smooth functions. To tackle this problem Gilpin et al. [23] smooth these functions using the excessive gap technique [53]. They then scale down the contribution of the smoothing functions as they approach the saddle point, eventually arriving at an ϵ-Nash equilibrium for the game. Gilpin et al. report that using this technique they were able to handle games that are several orders of magnitude larger than games that can be solved using conventional linear programming solvers [23].

3.4 Abstraction

When a game is too large to solve directly, a technique known as abstraction is often used. The idea is to create a small game with the same basic properties as the larger game and then solve the smaller game. The process of creating the smaller game is known as abstraction, and the smaller game is often called the abstract game. The solution to the small game is adapted to be used in the larger, original game. A successful abstraction technique is one that creates a smaller game that retains the important strategic features of the original game. The larger the game, the more difficult this process is. I will discuss abstraction of no-limit Texas hold'em in Section 3.8.2.

3.4.1 Outline of approach to solving extensive form games

The typical approach to solving large extensive form games it to first perform abstraction, creating a much smaller game. The abstract game is then solved using an equilibrium solver. Finally the strategy for the abstract game is converted into a strategy to play the full game in a process called translation. The translation problem is more complex when action abstraction and state abstraction are used together (see Section 3.8.2). This process is represented graphically in Figure 3.1.

3.5 Work in large or continuous action spaces

While considerable attention has been given to deriving an optimal policy for an agent in a large or continuous state space, little work has been done considering domains with large or continuous action spaces [47, 31]. The two strategies that have been pursued are to either create coarse-grain abstractions of the action space and apply standard methods for discrete action spaces [12, 48], or to sample from the continuous action space according to some distribution [47, 31]. In this section I discuss work outside of the domain of no-limit poker. I summarize research on creating no-limit agents in Section 3.8.

Lazaric et al. [47] tackle domains with continuous action spaces by using an actor-critic approach in which the policy of the actor is estimated using sequential Monte Carlo methods. They then use importance sampling on the basis of the values learned by the critic, and they apply resampling to modify the actor's policy. They show results where their algorithm performs better than static solutions such as continuous Q-learning [47]. While this approach had some success in static

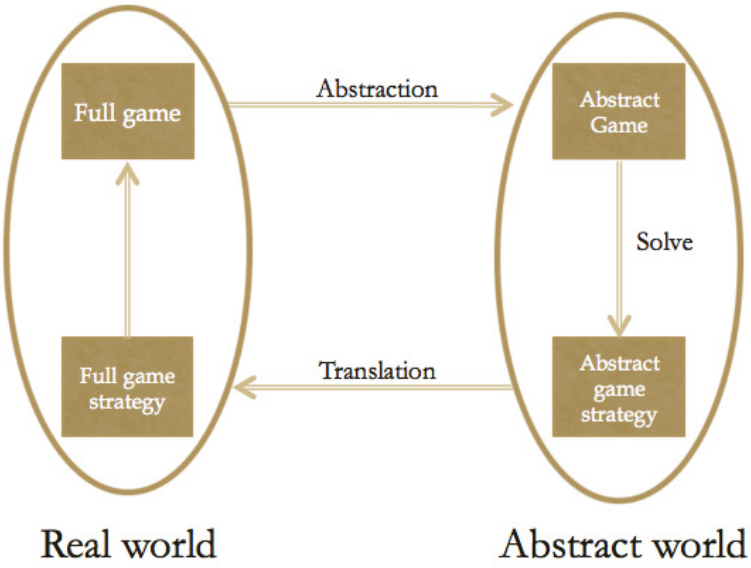


Figure 3.1: Typical process of solving a large extensive form game

domains, it is unlikely to scale well to domains with adversarial agents.[2] Hasselt et al. [31] also designed an actor-critic algorithm for continuous action spaces. The key difference between their algorithm and standard actor-critic algorithms is that they use the sign of the TD-error to determine the update to the actor instead of using the exact value. Their algorithm performs well in simple tracking and balancing problems [31].

Madeira et al. [48] developed very coarse-grained abstractions of the state and action space of large-scale strategy games, some of which have state spaces of size $\sim 10^{1900}$ and action spaces of size $\sim 10^{200}$. The resulting agents performed at an intermediate level on the Battleground simulator [48]. Similarly Dean et al. [12] create coarse-grain abstractions to compactly represent MDPs for planning problems with large state and action spaces. Their methods work most effectively in domains where, though there are a large number of actions at each state, only a few of these actions have much of an impact on achieving the final goal or minimizing costs [12].

Little theoretical work exists dealing with agents operating in domains with large or continuous action spaces. Nijmegen [64] shows that for infinite stage stochastic games with very large action spaces there exist ϵ-equilibrium points for all $\epsilon > 0$. Antos et al. [4] provide a rigorous analysis of a fitted Q-iteration algorithm for continuous action space MDPs, deriving what they believe to be the first finite-time bound for value-function based algorithms in continuous state and action space problems.

[2] Personal correspondence with Alessandro Lazaric, March 2009

3.6 The Annual Computer Poker Competition

Every year since 2006 the Annual Computer Poker Competition (ACPC) has been held in association with a major AI conference (AAAI or IJCAI) [32]. Initially the competition was played using limit Texas hold'em, and over time two more variants of Texas hold'em were added: limit with more than two players and two player no-limit. Starting in 2009 each game has been divided into two divisions, scored using different winner determination rules. The total bankroll division ranks the entries according to the total number of chips they win or lose against all other entries. This division is designed to emphasize opponent modeling: the amount by which an agent wins (or loses) against another agent is very important. The other division, known as bankroll instant run-off, is scored the same as the total bankroll division initially, then the worst performing competitor is removed from the tournament and the scores are recalculated. This process is repeated until there is only one competitor left. The idea of this scoring system is to try and measure which agents lose the least. The *amount* of chips won or lost is less important, and instead it is more important *if* an agent wins or loses against the other competitiors.

The problem of opponent modeling in poker is a very difficult one, and while the idea of the total bankroll division is to encourage research in this area, it has often been the case that the winners of this division did not have a modeling component. Recent work on modeling has shown promise — Bard et al. use an implicit modeling technique to choose between a set of pre-computed strategies [5]. They show that their implicit modeling agent would have won the 2011 two player limit total bankroll division of the ACPC had it been employed.

3.7 Limit Texas hold'em poker agents

Here I will outline the work that has been done on solving limit Texas hold'em poker.

3.7.1 Two player limit

In 2003 Billings et al. [7] developed the abstraction techniques necessary to make the game of two player limit Texas hold'em small enough to solve using linear programming algorithms. This allowed for the creation of the first agents competitive with world class players. The next major advancement was the development of the CFR algorithm by Zinkevich et al. [68], as described in Section 3.3. In 2007, agents created using the CFR algorithm were the first to compete head to head with world class professional poker players, losing by a small margin. Further advancements in state abstraction [67] improved the performance of these agents, leading to a victory over some of the best limit two player Texas hold'em players in the world in July 2008 [2].

Recently another algorithm has advanced the state of the art in two player poker. Building on their algorithmic technique for computing best responses quickly [41], Johanson et al. created CFR-BR, a CFR variant that is guaranteed to find, within a given state abstraction, the strategy

with the lowest exploitability in the full game [39]. Their algorithm uses a combination of chance-sampled CFR with best response calculations to achieve this goal. In another paper, Johanson et al. look at both improvements to state abstraction in poker games and proper evaluation of these abstractions [42]. Their results showed that using imperfect recall in state abstractions results in better performance, which is not surprising given the fact that many of the best entries in the limit two player divisions of the ACPC in the last few years have used imperfect recall in their state abstractions [32]. Additionally Johanson et al. show that effective state abstractions can be created by clustering together similar hands using the k-means algorithm [42].

Bots created using chance and action-sampling variants of the CFR algorithm have been the most successful in the ACPC, obtaining the most top three finishes [32]. The top two competitors in the limit two player bankroll instant run-off division of the ACPC in 2012 were CFR-based. Eric Jackson created the first place entry using a distributed disk-based implementation of chance-sampled CFR to run a small number of iterations (1440) on a very large card abstraction [36]. The second place entry was entered by Johanson et al., using CFR-BR applied to a state abstraction created using k-means clustering, as described earlier.

Approaches not using the CFR algorithm (or a variant thereof), while less common, have also enjoyed some success in the two player limit Texas hold'em domain. From 2009 to 2011 Marv Anderson created agents that won the two-player limit total bankroll competitions of the ACPC each year. These agents were created using a simple neural network, trained on winning entries from previous years [32]. Rubin and Watson use a case based reasoning approach trained on strong agents from previous years to generate agents, which has resulted in some top three finishes in recent years [55, 57]. Gilpin et al. [16, 20, 21, 22, 23, 24] have also had success creating limit two player agents using the gradient based approach discussed in Section 3.3. Much of their work has centered around developing techniques to automatically abstract the state space of the game [21, 22, 24]. Another of their techniques modifies the play of their agents by solving the river independently using a continuous approximation to Texas hold'em, resulting in an improvement in performance when tested against the agents entered in the 2008 two player limit division of the ACPC [16].

Much of what currently differentiates the best two player limit Texas hold'em agents revolves around work on state abstraction. It is important to note that Waugh et al. showed that simply increasing the size of an abstraction of a poker game does not necessarily lead to a better agent — in fact, the resulting agent can get worse [65, 66]. Nonetheless agents using larger abstractions have, in general, beaten agents with smaller abstractions, when the abstractions were generated using similar techniques.

3.7.2 Ring limit

From 1997 to 2001 Billings et al. worked to create Loki and its successor Poki to play multiplayer limit Texas hold'em, a very common game played in most casinos [6, 8]. They used a combination

of different approaches, including expert knowledge, Monte Carlo roll-out simulations, exhaustive enumeration algorithms to determine the value of different poker hands (hand strength) and some basic opponent modeling [8]. Poki was a substantial improvement over Loki, and was the first poker agent capable of beating average human players in low limit casino games [6]. The size of the multiplayer game proved to be too great to be able to develop world class poker agents using the technologies of the time. For this reason much of the academic poker research since the development of Poki has focused on two player poker, which I discussed above.

A return to research on ring limit started in 2008, when the ACPC added a single six player limit division. This was replaced with three player limit in 2009 and divided into two divisions, as described earlier. Sweeney et al. used a gradient-based approach to create static multiplayer agents for both the six player game in 2008 and the three player game in 2009 [63]. Park et al. applied Monte Carlo search to create agents, also applying their technique to both game types [60]. Abou-Risk et al. applied the CFR algorithm to a small abstraction of the three player game, obtaining a base strategy. They then created a new agent by picking certain two player sub-games (where one player had folded), solving them independently, then using them in place of the base three player strategy at these points [3]. The agents created by Abou-Risk et al. were the most successful of the time, placing first and second in both of the 2009 ACPC three player limit divisions [32]. They also significantly out perform Poki, the winner of the six player division of the ACPC in 2008, in three player limit [3].

Gibson et al. continued the work of Abou-Risk et al., dominating the three player limit division of the ACPC in 2010, 2011 and 2012 [32]. Gibson et al. looked at increasing abstraction granularity in very large games, such as three player limit hold'em, by partitioning the game tree into parts and computing strategies for each part independently, then combining the strategies in a process known as strategy stitching [19]. This led to the winning entries of the 2010 and 2011 three player events. A technique for computing these strategies concurrently with the rest of the game tree was used to create their 2012 competition entry, also leading to a victory in the three player events.[3] The other notably successful entry in the three player limit division in recent years comes from Rubin and Watson, again applying their case based reasoning approach mentioned above [57]. The resulting agents obtained multiple top three finishes in the limit three player division from 2010 to 2012 [32].

3.8 No-limit Texas hold'em poker agents

The work that has been done on no-limit Texas hold'em can be divided into two categories: solving games with a very small stack size to blind ratio, and solving games with a much larger ratio. These are known as short stack and deep stack games, respectively.

[3] Personal correspondence with Richard Gibson, April 2013

3.8.1 Short stack no-limit

Miltersen and Sorensen [50] derived a strategy for a two player no-limit Texas hold'em game where both players have stacks of 6.67 big blinds, a very small stack size. Their strategy involves only allowing the small blind to either fold or go all-in. This type of strategy is known in the poker community as a jam-fold strategy [9, 61]. They proved that this strategy is close to a Nash equilibrium in the full game (with this stack to blind ratio).

Sandholm and Ganzfried devised an algorithm that obtains a Nash equilibrium solution in three player no-limit Texas hold'em, using stacks of 7.5 big blinds for all three players [14, 15]. While their algorithm is not guaranteed to converge, they prove that if it does converge the resulting strategy profile is an ϵ-Nash equilibrium. This solution suffers from problems faced frequently in multiplayer game theory — while the strategy they obtain is an ϵ-Nash equilibrium, this only guarantees that if one player deviates from this strategy profile the other two will not lose value. If two players simultaneously deviate from this strategy profile then the value of the third player's strategy may be reduced. Additionally, there may be many Nash equilibria, with different values for each player — it may be possible for two players to implicitly coordinate their strategy to the detriment of the third player. Such a strategy profile may be unrealistically pessimistic for some players and overly optimistic for others [9]. Sandholm and Ganzfried state that they expect there is a unique equilibrium, negating the former problem, and that they doubt the latter problem is significant when players are playing jam-fold strategies [14].

3.8.2 Deep stack no-limit

Work on deep stack no-limit Texas hold'em (stacks of 100 big blinds or more) is much more complex, due to the larger game size. In limit poker it is not necessary to abstract the action space, and in short stack no-limit a simple jam-fold abstraction is sufficient to represent strategies that are close to Nash equilibria in the full game, as mentioned above. In deep stack no-limit, abstraction of the action space is required if the techniques described in Section 3.3 are to be used. This adds two challenges to solving the game — creating a reasonable action abstraction, and designing an algorithm to translate any information set in the full game to an information set in this abstraction. The focus of my thesis is to provide an answer to the former problem. The latter problem is known as the translation problem, and significant work on this has been done by Schnizlein et al. [58, 59], which I discuss below.

Choice of action abstraction

Typical abstractions of the no-limit action space allow fold, call and all-in actions everywhere, but limit the number of different sized bets players can make. The allowed bets are usually expressed in terms of the pot size. Common bets include half pot, pot, and integer multiples of the pot.

Gilpin et al. created no-limit agents to play the 200 and 500 big blind variants using their

technology developed for limit [25]. As an abstraction of the action space they allowed fold, call, pot and all-in everywhere, along with half pot at select places in the game. Their reasons for choosing this abstraction were based in established poker theory [25].

Schnizlein created no-limit agents using the CFR algorithm and a variety of abstractions to play the 100, 200 and 500 big blind games [58]. In addition to fold, call, pot and all-in, some of the bet sizes he considered were half pot, quarter pot, 1.5 pot, 2 pot, 10 pot and 11 pot. He did experiments keeping the size of the game constant and alternating between allocating more space to the card abstraction and the action abstraction. As was discussed in Section 1.3 his results showed that agents using small bet options such as half pot outperformed agents of the same size with simpler betting abstractions but much more complex card abstractions [58].

Betting translation

Typically when a no-limit game is solved using an abstraction of the action space, this is done with the intention of using the resulting strategy to play the full no-limit game. In order to do this we must decide how we will handle actions made by the opponent that are not in our abstraction of the game. There are multiple algorithms that can be used to map actions in the real game to actions in the abstracted game. The simplest of these, hard translation, maps each action to the closest action in the abstraction using a predefined metric. The most successful metric is known as the geometric metric. If b is the real bet size and c and d are the closest bet sizes b in the abstract game such that $c < b < d$, then we map b to c if $c/b > b/d$ and we map to d otherwise [25, 58].

An alternative to hard translation, known as soft translation, again maps a real bet size b to the closest abstract bet sizes. While hard translation simply maps to one of the abstract actions, soft translation maps to both of them with some probability. Schnizlein et al. [58, 59] map to the lower bet size with probability $P = \frac{c/b - c/d}{1 - c/d}$ and the upper bet size with probability $1 - P$, where b, c and d are defined as above. Ganzfried et al. perform soft translation using different probabilities - in this example they map to c with probability $P = \frac{cd}{b^2 + cd}$, and d with probability $1 - P$.[4] Soft translation has been shown to outperform hard translation [58].

In the 2009 ACPC an adaptive no-limit poker agent designed to exploit the weakness of existing translation techniques was entered by Schnizlein [58] and dominated the two player no-limit competition. It beat some equilibrium-based agents by over 2 small blinds per hand, more than they would lose if they simply folded every hand [32]. This is a compelling demonstration of the difficulty of the translation problem.

In 2010 and 2011 the no-limit bankroll instant run-off division of the ACPC was won by agents generated by applying variants of the CFR algorithm to hand-crafted action abstractions, and then using translation of the form discussed above [32]. In the 2012 competition the action abstraction used by the winning entry in the no-limit bankroll instant run-off division was generated by the

[4] Personal correspondence with Sam Ganzfried, February 2009

author using the techniques described in this dissertation (see Section 5.9).

Chapter 4

Solving for bet sizes in small games

In this chapter I introduce a transformation that can be applied to imperfect information extensive-form games in which one or more actions at many decision points have an associated continuous or many-valued parameter. Specifically, I show this transformation as it applies to no-limit poker games. Section 4.1 introduces the transformation, which creates a new multi-agent game consisting of two teams, where each team has a single decision agent and possibly multiple parameter-setting agents, one for each parameter. In Section 4.2, I show that a known Nash equilibrium solution for a small no-limit poker games maps to a Nash equilibrium solution of the transformed game. Section 4.3 introduces a new regret-minimizing algorithm, called $BETS$, that can be applied to find equilibria of the transformed game. In Section 4.4 I show that by performing this transformation and using the $BETS$ algorithm I can determine the bet-sizes made as part of a Nash equilibrium solution of the original game. Finally, Section 4.5 introduces the local maxima problem encountered when using this technique, which I consider in more detail in Chapter 6.

The core material in this chapter has been published in AAAI 2011 [33].

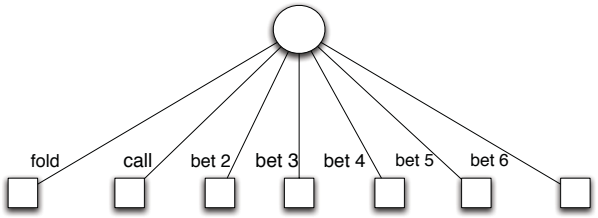
4.1 The multiplayer betting transformation

I now define the multiplayer betting transformation. It can be applied to domains with actions that have associated real or multi-valued parameters. In the case of poker, the action in question is a bet, and the multi-valued parameter is the size of the bet. These are combined to create a large number of possible actions of the form "bet x", where x is an integer between some minimum value and the stack size of the betting player. In the case of a trading agent, each of the actions buy and sell could have a parameter that specifies the amount. As we use poker games as our test bed, I will use poker terminology throughout this dissertation.

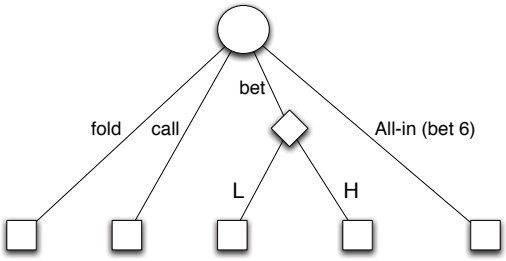
The transformation is applied to a two-player no-limit poker game as follows. At each decision node with m betting actions such that $m > 1$, remove all betting actions except all-in. Insert some predefined number q of betting actions such that $q < m$. A different value of q may be used at each decision node. Each new bet action is followed by a separate decision node for a new player, who I

will refer to as a "bet sizing player". This player chooses between a low bet and a high bet, which I will refer to as betting L or betting H, where these are expressed as fractions of a pot sized bet. Every bet action is assigned a new bet sizing player. The low and high bets can be different for each bet sizing player, with the only restrictions being that for bet sizing player i $L_i < H_i$, L_i is at least a minimum bet and H_i is at most an all-in bet. This means that if bet sizing players i and j branch from the same decision node, they may each have L and H values such that $[L_i, H_i]$ overlaps $[L_j, H_j]$.

The new bet sizing players do not see any of the private cards, and each bet sizing player obtains the same payoffs as the player for whom they are choosing the bet. So instead of two players, the game has two teams. The decision of whether to bet L or H is private to the bet sizing player that chose it — no other player in the game observes this action. I will refer to the two players that have fold, call, bet and all-in actions as player one and player two, and I will refer to the game that is created when this transformation is applied to game X as "multiplayer betting X". Figure 4.1(a) shows a decision node for player one in a no-limit game with a 4 chip pot, 6 chips left in each stack and a minimum bet of 2. The corresponding decision node for the multiplayer betting version of this game, with $q = 1$, is shown in Figure 4.1(b). Here circles represent player one, squares represent player two, the diamond is a bet sizing player, and any values of L and H can be chosen such that $L \geq 0.5$ and $L < H \leq 1.5$.



(a) Regular no-limit game



(b) Multiplayer betting game

Figure 4.1: A decision node in a no-limit game, and the multiplayer betting version of the same decision node with $q = 1$.

I now define the effective bet size function $B(s)$, where $P(L)$ is the probability of making the low bet L and $P(H)$ is the probability of making the high bet H.

$$B(P(H)) = (1 - P(H))L + P(H)H. \tag{4.1}$$

Here I used $P(H) + P(L) = 1$ to eliminate $P(L)$.

Lemma 1. *For any fixed values of L and H and probability $P(H)_i$ of player i betting H, the expected value of the pot size after that bet is equal to $p + B(P(H)_i)p$, where p is the pot size before the bet.*

Proof. After a bet of size L the pot size is $p + pL$, and after a bet of size H the pot size is $p + pH$. So the expected pot size is $P(L)_i(p + pL) + P(H)_i(p + pH) = P(L)_i p + P(H)_i p + P(L)_i pL + P(H)_i pH$. Since $P(L)_i + P(H)_i = 1$, this is equivalent to $p + ((1 - P(H)_i)L + P(H)_i H)\, p$, and using Equation 4.1 we get $p + B(P(H)_i)p$, as required. □

This leads me to define the *equivalent abstract game*

Definition 1. *Given a no-limit poker game* G*, a transformed game* G′ *obtained using the multiplayer betting transformation, and a strategy profile σ for* G′*, I define the equivalent abstract game as an action abstraction of* G *created by taking* G′ *and replacing each bet sizing player i by a bet of size $s_i = B(P(H)_i)$.*

Finally, I present the following theorem.

Theorem 1. *For any given strategy profile σ of a multiplayer betting game, $u_i(\sigma)$ where $i = 1$ or $i = 2$ is equivalent to $u_i(\sigma')$ in the equivalent abstract game, and $\sigma'_i = \sigma_i$ for $i = 1, 2$.*

Proof. For every terminal history z of the equivalent abstract game involving k betting actions there is a corresponding set Z of 2^k terminal histories in the multiplayer betting game where players one and two make the same actions. The bet sizes in the equivalent abstract game correspond to the probabilities of choosing L or H, so, by Lemma 1, the pot size for history z is equal to the expected pot size for player one and player two for Z. Therefore, the utility of z equals the expected utility for all of Z. Since the probability of z is equal to the combined probability of histories in Z, the utility of a strategy in the abstract game is equal to the utility of the corresponding strategy in the multiplayer game for players one and two. □

4.2 Transformation significance

Theorem 1 allows us to map a given strategy profile σ for a multiplayer betting game to a strategy profile σ' for the equivalent abstract game with the strategy profiles having the same value in both games. Each different choice of $P(H)_i$ for each bet sizing player i corresponds to a different equivalent abstract game, making the choice of abstraction part of each team's strategy in the multiplayer betting game. Additionally, a strategy profile for the multiplayer betting game and a strategy profile for the equivalent abstract game require almost the same memory to store, with the multiplayer

betting game defining exactly q extra parameters, where q is the number of bet-sizing players. Any abstraction of the full no-limit game can be represented by a strategy in a multiplayer betting game given appropriate values of L, H and q. Thus with careful choices of these parameters (a topic I cover in Chapters 5 and 6), we can represent all abstractions that we are interested in using, while requiring only slightly more memory than any one of those abstractions to store the strategy.

This is not the only value of the multiplayer betting transformation. As I demonstrate below, in small poker games the bet sizes used in Nash equilibrium strategy profiles correspond to the effective bet sizes used in Nash equilibria of the multiplayer betting game. This suggests that a Nash equilibrium of the multiplayer betting game can be mapped to an excellent choice of abstraction in the full poker game. It should be noted here that the equivalent abstract game, as given by Definition 1, is a game where any real valued number of chips can be bet, while in reality poker games (with the exception of some toy games) are played using integer bet sizes. In practice we can round to the nearest integer, and because in poker the pot grows exponentially with each bet [62], this is a reasonable approximation.

4.2.1 Analysis in Kuhn poker

Here I will consider the properties of a Nash equilibrium solution of multiplayer betting no-limit half street Kuhn poker. To do this I use analytical formulas previously derived for half street Kuhn poker with fixed bet sizes that relate Nash equilibrium strategies to the bet size used [9].

In half street Kuhn poker the first player has two possible actions: bet or check. If the first player bets, the second player must either fold or call, while if the first player checks the game ends (see Section 2.2.1). In a Nash equilibrium strategy profile the first player always bets with the the King and never bets with the Queen, while the second player always calls with the King and never calls with the Jack [9]. Thus the game has two non-dominated strategy parameters, $P_1(b \mid j)$, the probability that the first player bets with the Jack, and $P_2(c \mid q)$, the probability that the second player calls with the Queen. For any version of this game with antes of 0.5 and any number of real-valued bet sizes s_i such that $0 < s_i < 1$, a strategy profile is a Nash equilibrium if and only if, for all bet sizes s_i such that $P_1(b \mid j)_i \neq 0$ [9],

$$P_1(b \mid j)_i = \frac{1 - P_2(c \mid q)_i}{2} = \frac{s_i}{s_i + 1}. \tag{4.2}$$

Note that if $s_i > 1$, then in any Nash equilibrium strategy profile $P_1(b \mid j)_i = 0$, $P_2(c \mid q)_i = 0$ and this subgame becomes trivial — the first player only considers betting s_i with the King, and the second player only calls a bet of size s_i with the King.

Consider a multiplayer betting version of this game. All bets in half street Kuhn poker can be represented by a bet-sizing player using an equivalent effective bet size $B(P(H))$. To find a Nash equilibrium for multiplayer betting half street Kuhn poker we must obey Equation 4.2 where $s_i = B(P(H)_i)$ for every bet-sizing player i, and we must also pick $P(H)_i$ such that player i

cannot gain by choosing a different value. For this to be true we must have, for all $0 \leq x \leq 1$, $u_1(B(x)) \leq u_1(s)$ and thus $u_1(B(x)) - u_1(s) \leq 0$. The only situations that affect this difference in utility for a given strategy profile σ are sequences that involve a bet being made and called. Thus we consider the probability player one bets with the King and is called (winning the hand), minus the probability player one bets with the Jack and is called (losing the hand). So we have

$$u_1(B(x)) - u_1(s) = \left[\frac{P_1(b \mid k)}{3}\frac{P_2(c \mid q)}{2} - \left(\frac{P_1(b \mid j)}{3}\left(\frac{P_2(c \mid k) + P_2(c \mid q)}{2}\right)\right)\right]\Big(B(x) - s\Big). \tag{4.3}$$

It is dominated to check or fold the King so $P_1(b \mid k) = 1$ and $P_2(c \mid k) = 1$ and we have

$$u_1(B(x)) - u_1(s) = \left[\frac{P_2(c \mid q)}{6} - \frac{P_1(b \mid j)}{3}\left(\frac{1 + P_2(c \mid q)}{2}\right)\right]\Big(B(x) - s\Big) \leq 0. \tag{4.4}$$

For this to be true for any $B(x)$ we must have

$$P_1(b \mid j) = \frac{P_2(c \mid q)}{1 + P_2(c \mid q)}. \tag{4.5}$$

To have a Nash equilibrium in the multiplayer betting half street Kuhn poker game we need to satisfy Equation 4.5 and Equation 4.2, so

$$\frac{1 - P_2(c \mid q)}{2} = \frac{P_2(c \mid q)}{1 + P_2(c \mid q)}, \tag{4.6}$$

a quadratic to which the positive solution is

$$P_2(c \mid q) = \sqrt{2} - 1. \tag{4.7}$$

Plugging this into Equation 4.2 and solving for s we get

$$s = \sqrt{2} - 1. \tag{4.8}$$

So a Nash equilibrium strategy profile in the multiplayer betting half street Kuhn game must define all values of $P(H)_i$ where $L_i < H_i < 1$ such that $B(P(H)_i) = s = \sqrt{2} - 1$.

Chen and Ankenman also show that the value of half street Kuhn poker to the betting player as a function of bet size s is given by [9]

$$V(s)_1 = \frac{s(1 - s)}{6(1 + s)} \tag{4.9}$$

Figure 4.2 shows a plot of this function from $s = 0$ to $s = 1$. Each point on this curve represents the game value for a given bet size s. For every point on the x-axis the Nash equilibrium strategy profile may be completely different.

Chen and Ankenman show that the curve peaks at $s = \sqrt{2} - 1$, thus the Nash equilibrium in no-limit half street Kuhn poker uses this bet size. This is the same as Equation 4.8, so if we can find a Nash equilibrium for any multiplayer betting half street Kuhn poker game where we set

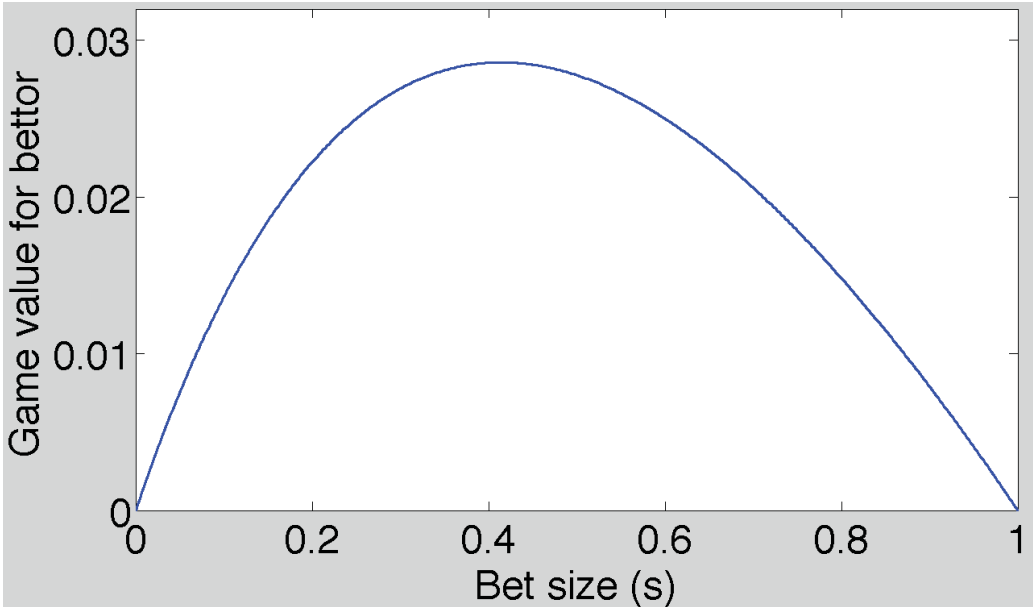


Figure 4.2: Half-street Kuhn poker game value per bet size.

$L_i < \sqrt{2} - 1 < H_i < 1$ for all bet-sizing players i, then the value of the equivalent abstract game for player one will be the same as the value of a Nash equilibrium strategy profile of the full no-limit game for player one. Note that the Nash equilibrium of no-limit half street Kuhn poker uses only one bet size, so using multiple bet-sizing players in the transformation is unnecessary (for this game), as in a Nash equilibrium strategy profile they will all set $P(H)_i$ to the same effective bet size.

4.3 Computing multiplayer betting games strategies

The multiplayer betting transformation creates an n player game, where $n > 2$. Unfortunately this means that the standard techniques for minimizing regret in two-player games are no longer guaranteed to converge. This does not mean it is not worth trying — Gibson et al. [19] have had success applying variants of the CFR algorithm to three player poker games, as discussed in Section 3.7.2. Unfortunately, when I applied chance-sampling CFR [37] directly to the multiplayer betting version of some small poker games, it did not converge. Analysis of these runs shows that the bet-sizing player mostly alternates between having positive regret for one action and negative for the other, switching between playing pure strategies $P(H) = 1$ and $P(H) = 0$. Looking at Equation 4.2 we can see that in a Nash equilibrium strategy profile for no-limit half street Kuhn poker $P_1(b \mid j)_i$ and $P_2(c \mid j)_i$ are not quite linear in s_i. With this in mind I modified the chance-sampled CFR algorithm as follows: I continue to use regret matching to minimize regret for players one and two (see Section 3.3.2), but for each bet-sizing player I move each effective bet size $s_i^t = B(P(H)_i^t)$

small amounts in the direction that minimizes regret for player i:

$$s_i^{t+1} = \begin{cases} \frac{(t-1)s_i^t + sr_i^{t+1}}{t} & \text{if } t < N \\ \frac{(N-1)s_i^t + sr_i^{t+1}}{N} & \text{otherwise.} \end{cases} \tag{4.10}$$

where N is a parameter chosen beforehand, s_i^t is the effective bet size made by bet-sizing agent i on iteration t defined as $B(P(H)_i)$ from Equation 4.1, and sr_i^{t+1} is the effective bet size that would be used on iteration $t+1$ if I was using regret matching. We can see that up to iteration $t = N$ the effective bet size chosen by this algorithm is simply the average of all the effective bet sizes that regret matching would pick each iteration. Using this average makes the movement of the effective bet size from iteration to iteration slower and more smooth. After time $t = N$ we still use the average of all past choices of regret matching, but we weight the effective bet size of the latest choice by $1/N$. This ensures that the effective bet size continues to move some minimal amount in a direction that minimizes regret.

When this new regret minimizing algorithm is applied to half street no-limit Kuhn poker using $L = 0$, $H = 1$, $N > T$ and run for $T = 500,000$ iterations, the average of all $P(H)_i^t$ choices made by bet sizing player i converges to within 1% of $B(\overline{P(H)}_i^T) = 0.414$. Note that this is the same value as Equation 4.8.[1] The algorithm, however, is significantly slower than CFR. In CFR, the average (not the current) action probabilities converge to an ϵ-Nash equilibrium. Therefore, for each action sequence the algorithm maintains a sum of the probability this sequence was played at each iteration. Any time a player has 0% probability of reaching an information set, the addition step can be skipped for all action sequences that reach the subtree beneath that information set. This cutoff can always be used, independent of how the bet-sizing players are updated. This leads to a new update equation:

$$s_i^{t+1} = \begin{cases} s_i^t + \frac{B_i^t(sr_i^{t+1} - s_i^t)}{t} & \text{if } t < N \\ s_i^t + \frac{B_i^t(sr_i^{t+1} - s_i^t)}{N} & \text{otherwise} \end{cases} \tag{4.11}$$

Here B_i^t is the probability all players play to reach this information set on iteration t. Equation 4.10 updates s_i^t for each bet sizing player on every iteration. If $B_i^t = 0$ then Equation 4.11 implies that $s_i^{t+1} = s_i^t$. In this case, the algorithm can make a single strategy update pass for each team of bet-sizing players at the same time that it updates the totals used to calculate the average strategy of the corresponding main player. The resulting algorithm, again with $L = 0$, $H = 1$ and $N > T$, also converges to $B(\overline{P(H)}_i^T) = 0.414$, but does so significantly faster than when Equation 4.10 is used.

I call the process of minimizing regret in a multiplayer betting game using the result to obtain an abstraction Bet-size Extraction through Transformation Solving, or $BETS$, and it is outlined in Algorithm 1.

While the choice of $N > T$ leads to convergence in half street Kuhn poker, in larger Kuhn games $N > T$ no longer converges. I found that a setting of $N = 10000$ for all bet sizing players worked

[1] This was tested with one, two and three bet-sizing players

Algorithm 1 The $BETS$ algorithm

Require: Multiplayer betting game G'
Require: T iterations
  Apply modified chance-sampled CFR to G' as follows:
  for t from 1 to T **do**
    Calculate regrets as in CFR
    During strategy-update phase:
    for each player i **do**
      if i is a bet-sizing player **then**
        $P(H)_i^{t+1} = s_i^{t+1}$ from Equation 4.11
      else
        Set σ_i as in CFR
      end if
    end for
  end for
  Create abstraction A, replacing each bet sizing player i in G' by a bet of size $s_i = B(\overline{P(H)}_i^T)$
  return A

well in all test cases, and this is the value used to produce all of the results presented throughout the dissertation.

4.4 Computing abstractions for no-limit Kuhn poker

Applying $BETS$ I computed abstractions for three variants of no-limit Kuhn poker for which the Nash equilibria have been determined analytically [9]. The results are shown in Table 4.1, and detailed below.

Kuhn Variant	Bet sizes used by Nash equilibrium	Bet sizes found by $BETS$ algorithm
Half street	0.414	0.414
1 bet cap	0.509 and 0.414	0.509 and 0.414
Unlimited raises	0.252, 0.542 and minimum raise	0.252, 0.542 and L for all L, H tried

Table 4.1: Bet sizes used in Nash equilibria of various Kuhn poker variants, with corresponding bet size found by applying $BETS$.

For multiplayer betting half street Kuhn poker with $L = 0$ and $H = 1$, a $500,000$ iteration run of $BETS$ converges to within 1% of $B(P(H)) = 0.414$, the bet size used by the Nash equilibrium strategy profile of the full no-limit game [9]. For cases where $H < 0.414$ the algorithm converges with $P(H) = 1$. Figure 4.2 shows that this is the bet size with the highest game value for the betting player in that range. For choices of $L > 0.414$ with $H \leq 1$, $BETS$ converges to $P(H) = 0$, again obtaining the maximum game value for the given range (for a discussion of what happens when $H > 1$ see Section 4.5).

In the larger game of no-limit Kuhn poker where each player may make a bet but not raise, the bet sizes used by Nash equilibrium strategies are $s = 0.414$ for player one and $s = 0.509$ for player two [9]. Here a one million iteration run of $BETS$ converges to within 1% of the effective

bet sizes of $B(P(H)) = 0.414$ and $B(P(H)) = 0.509$ for teams one and two respectively. In the full no-limit Kuhn poker game both players may bet and raise indefinitely. The bet sizes used by a Nash equilibrium strategy profile in this game are quite unintuitive, like the previous games. The bet sizes are 0.252 for player one's initial bet, 0.542 for player two's bet if player one checks, and the minimum legal raise if player two raises in response to a player one bet [9]. Using $L = 0.1$ and $H = 0.3$ for player one's bet and $L = 0.4$ and $H = 0.6$ for player two's first bet $BETS$ converges to within 1% of the 0.252 and 0.542 bet sizes after two million iterations, and for all ranges that I tried for the player two raise, $BETS$ sets $P(H) = 0$.

4.5 Local maxima

In Section 4.2 I showed that in no-limit half street Kuhn poker the Nash equilibrium bet size corresponds to the strategy used by the bet-sizing player in the Nash equilibrium of a multiplayer betting game where $L < 0.414$ and $H > 0.414$. The reverse is not always true — games with more than 2 players may have many equilibria with different game values, and the multiplayer betting game is no exception to this. An example of this can be seen in half street Kuhn poker, when $H > 1$. To understand what happens at this point, I extend the game value curve from Figure 4.2 for larger bet sizes, producing Figure 4.3.

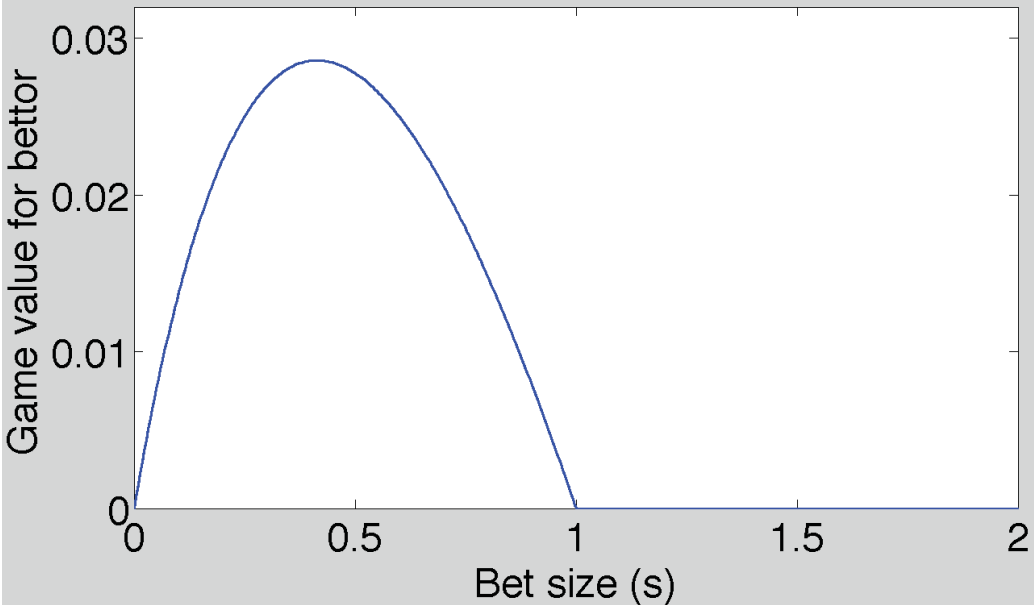


Figure 4.3: Half-street Kuhn poker game value per bet size, extended to 2 pot.

Examining this figure you can see that for bets of size $s > 1$ the value of the game is 0. For all of these bet sizes the strategies for both of the main players are the same - there is no longer any bluffing with the Jack or calling with the Queen. The first player only bets with the King, and the second player only calls with the King. While these points do not represent Nash equilibria of the

full no-limit game, they are Nash equilibria in the multiplayer betting game. This is due to the fact that while player one and the betting player could gain value if they both changed their strategy to the peak point where $s = 0.414$, they cannot gain value if just one of them changes their strategy. This means that for any multiplayer betting version of no-limit half street Kuhn poker where $H > 1$, $BETS$ may converge to an effective bet size $s \geq 1$, even if $L < 1$ (and indeed in some cases I observed this behaviour). I will return to this discussion in Chapter 6.

In the next chapter I move on to discuss the application of this transformation and algorithm to larger games, the complications that arise, and how I resolve them.

Chapter 5

Dynamically Adjusted Bet-Sizing Ranges

In this chapter I consider the application of the $BETS$ algorithm (see Section 4.3) to large poker games, specifically Leduc and Texas hold'em. Action abstractions in these games must be evaluated differently, which I discuss in Section 5.1. Sections 5.2 and 5.3 show that the application of $BETS$ to these games leads to an improvement in game value, however it is also evident that the static bet size ranges are limiting. To investigate this limitation, I introduce a metric for ranking bets in order of how much value could be gained by changing their range (Section 5.4). Next in Section 5.5 I apply this metric to the results of Sections 5.2 and 5.3, clearly demonstrating the potential gains of adjusting the ranges. Section 5.6 shows that it is preferable to make these ranges as small as possible. The desire for small, dynamic ranges led me to the RE–$BETS$ algorithm, which I introduce in Section 5.7. I apply RE–$BETS$ to Leduc and Texas hold'em in Section 5.8, leading to a significant improvement over the $BETS$ algorithm. Finally, in Section 5.9 I describe how RE–$BETS$ was used to create the winning entry in the 2012 ACPC no-limit bankroll instant run-off division.

Note that in this chapter I use one bet-sizing player at each decision node when performing the multiplayer betting transformation (see Section 4.1). I consider multiple bet-sizing players per decision node in Chapter 6.

The core material in this chapter has been published in AAAI 2012 [34].

5.1 Evaluation of action abstractions

The results in the previous chapter are from small games for which analytical solutions are known, providing a gold standard for evaluating my algorithm. For larger games such as no-limit Leduc poker and no-limit Texas hold'em poker I have neither an analytical expression for the Nash equilibrium, nor can I solve the full no-limit game using modern techniques. Instead, I compare the game value of different choices of action abstraction for one of the players against a fixed choice

(fold, call, pot and all-in in this chapter) for the other player. I call the fixed action abstraction the "opponent action abstraction".

Some of the action abstractions used in the comparisons I make in this chapter are generated by the $BETS$ algorithm from Chapter 4 and the RE–$BETS$ algorithm (see Section 5.7), while others are hand-crafted. I use the same card abstraction when making direct comparisons — for the analysis presented in this chapter (excluding Section 5.9) I use a card abstraction of no-limit Texas hold'em created using the hand strength metric [65] with 5 buckets each round and perfect recall. This leads to an abstraction with 5 buckets on the pre-flop, 25 on the flop, 125 on the turn and 625 on the river. No card abstraction is used in my no-limit Leduc experiments.

I compute the strategies that I use for my comparisons in two steps. The first step involves running $BETS$ (or later RE–$BETS$) which creates a new action abstraction. After this I run a chance-sampled version of the CFR algorithm [68], which adjusts only the strategy parameters, using the bet sizes that were computed in step one. The CFR algorithm produces an ϵ-Nash equilibrium for this new action abstraction, and thus calculating a best response to this strategy profile gives me bounds on the game value of the new action abstraction.[1] Note that this evaluation technique is for perfect recall card abstractions. Evaluating action abstractions that are paired with imperfect recall card abstractions is discussed in Chapter 7.

5.2 Leduc poker with limited raising

I consider a Leduc poker game where each player has 400 chips, with antes of 1 chip each. This game has approximately 10^{11} states and 10^{11} action sequences. Due to the large number of action sequences, it is difficult to solve the game directly with an ϵ-Nash equilibrium solver, as it would require at least 1480 gigabytes of memory. Instead I applied my methodology for evaluating large games as follows. I created a baseline abstraction where both players use an action abstraction with actions fold, call, pot and all-in (known as the $FCPA$ abstraction), against which I can compare the game value of each equivalent abstract game. I then created two multiplayer betting games, each time replacing each of the pot bets for one of the players with a single bet sizing player with $L = 0.8$ and $H = 1.5$. I call the player with the bet-sizing teammates the main player, and the other player the opponent, a terminology I use throughout the rest of the dissertation. I allowed a maximum of two raises per round for both the baseline Leduc game and the multiplayer betting games, a restriction which I explain in Section 5.8.

I applied the $BETS$ algorithm (Section 4.3) to each of the multiplayer betting games, running for 15 million iterations each time. Once this completed, I applied chance-sampled CFR to the equivalent abstract games, running for one billion iterations to assure a very small ϵ. Getting best response values for the resulting strategy I obtained game values of -0.060 antes for player one's

[1] I use the term "best response" throughout the dissertation to refer to a best response within the abstract game, as opposed to a best response in the full game

new betting abstraction and 0.078 for player two's new betting abstraction. Running chance-sampled CFR for one billion iterations on the baseline $FCPA$ abstraction and getting best response values I obtained a game value of -0.072 antes for player one (and an equivalent 0.072 antes for player two). The abstractions chosen by my algorithm for player one and two represent respective 16.7% and 8.3% improvements over the baseline action abstraction. We revisit these results in Section 5.8.

5.3 Application to Texas hold'em

For my application to no-limit Texas hold'em I used 200 big blind stacks and blinds of 50 and 100 chips. This is the variant of no-limit Texas hold'em played in the ACPC since 2009 (see Section 3.6). Once again I used $FCPA$ as the opponent action abstraction as well as the baseline abstraction, and I used the 5 bucket perfect recall card abstraction described in Section 5.1. I consider only player one in this section - I will look at both players one and two in Section 5.8.1.

Choosing $L = 0.8$ and $H = 1$ (suggested by experts) for all bet sizing players, I applied $BETS$ to player one, running for 200 million iterations, and I obtained an action abstraction. Next I ran chance-sampled CFR for one billion iterations on both this abstraction and the $FCPA$ baseline abstraction. Calculating the best responses of both abstractions, I found that the game value of the equivalent abstract game I generated for player one using $BETS$ was greater than that of the baseline action abstraction's value for player one by $\approx 17\%$.

I also created action abstractions for player one based on a number of shorter runs of $BETS$. Table 5.1 show the number of bet sizes in the given range after different length runs. This table shows that the vast majority of bet sizes are < 0.82, which is surprising, given that previous expert knowledge dictated that if only a single bet size is used everywhere, it should be pot sized.[2] So many bet sizes being close to the low boundary after only 1 million iterations suggests that the ranges should be moved down. It's possible, however, that the bets with these small values have little effect on the game value, while the more valuable bets remain closer to 1. This could happen, for example, if the bet-sizing players choosing these smaller bets were reached with very small probability. To test if this was the case I developed a metric, $R^T_{i,|\cdot|0}$, designed to rank the individual bet-sizing players in order of how much value could be gained by moving their range. I discuss this metric in the next section.

5.4 Importance metric

Here I define a metric that can be used to rank the bet-sizing players in a multiplayer betting game using two criteria. This metric, $R^T_{i,|\cdot|0}$, is designed to be calculated during a run of T iterations of the $BETS$ algorithm (see Section 4.3). First, it gives an indication of the importance of the sub-tree below bet-sizing player i to the overall game value. This is indicated by the probability that player

[2] Personal correspondence with Darse Billings, October 2010

Length	Bets $0.8 - 0.82$	Bets $0.82 - 0.98$	Bets $0.98 - 1.0$
1m	152	8	0
4m	151	9	0
6m	150	9	1
8m	145	13	2
10m	142	16	2
20m	138	20	2
100m	126	31	3
200m	126	28	6

Table 5.1: Bet sizes after different length runs of the $BETS$ algorithm.

i will be reached on a given iteration t where all players play according to σ^t, denoted by $\pi_{-i}^{\sigma^t}$, combined with the expected utility to player i if bet size s_i^t is played, given by $u_i^t(s_i^t, \sigma^t)$. Second, a difference calculation gives an indication of how much utility could be gained by moving player i's bet in the direction of positive regret on iteration t. If this value is high and the effective bet size is already at the end of the range, this indicates value could be gained by adjusting player i's range to allow further movement in the same direction.

The metric is calculated every iteration t, and summed over all iterations T. If no movement of a bet-sizing player's range is indicated, this metric can be used to sort the bet-sizing players in order of importance to the game value. If, however, there are bet-sizing players that would benefit from adjusted ranges, their score is increased. Thus by combining $R^T_{i,|\cdot|0}$ with an evaluation of how close the effective bet size chosen by each bet-sizing player is to the edge of its range, I can get an indication of which bet-sizing players are likely to provide the greatest increase in value if their range is adjusted.

On each iteration t of the $BETS$ algorithm, the bet-sizing players minimize immediate counterfactual regret [68]. To do this I compute two values:

$$R(L)_i^t = \pi_{-i}^{\sigma^t}\Big(u_i^t(L, \sigma^t) - u_i^t(s_i^t, \sigma^t)\Big) \tag{5.1}$$

and

$$R(H)_i^t = \pi_{-i}^{\sigma^t}\Big(u_i^t(H, \sigma^t) - u_i^t(s_i^t, \sigma^t)\Big), \tag{5.2}$$

where $s_i^t = B(P(H)_i^t)$ (see Equation 4.1). If increasing s_i^t would gain value, then $R(H)_i^t > 0$ and $R(L)_i^t < 0$. The opposite is true if decreasing s_i^t gains value. Clearly $R(H)_i^t$ tells us the value that can be gained (or lost) by maintaining the current strategy and increasing s_i^t. The opposite is true for $R(L)_i^t$. These regrets, however, only tell us how much value could be gained by moving the effective bet size to the edge of the range — if $s_i^t = H_i$,[3] then $R(H) = 0$. In such cases it would be nice to know if value could be gained by increasing the range.

Since all other probabilities are fixed during regret computation, $R(H)_i^t$ is linear in s_i^t. This is true because the amount of money that is won or lost is directly proportional to the utility u_i^t, which

[3]Note that L_i and H_i are the bet sizes made by bet-sizing player i, while L and H are the actions themselves

is linear in s_i^t. Therefore, the magnitudes of these values are proportional to the distance of s_i^t from the range boundaries. For example, as s_i^t gets closer to H_i, $|R(L)_i^t|$ increases and $|R(H)_i^t|$ decreases proportionally. This means that if $s_i^t = H_i$ and a lot of value would be gained if the bet size was increased further, then while $R(H)_i^t = 0$, $R(L)_i^t$ will be large and negative. For this reason I define

$$R_{i,|\cdot|}^t = |R(L)_i^t| + |R(H)_i^t|. \tag{5.3}$$

By combining the absolute values of these regrets, I get an indication of the value that could have been gained on iteration t by moving bet i in the direction of positive regret, independent of where s_i^t is in the range. In fact, $R_{i,|\cdot|}^t$ doesn't even depend on the values of L_i and H_i, only on $w = H_i - L_i$. I will prove this statement formally in the next section — see Equation 5.14.

Finally, I sum this value over all iterations t to obtain a metric for the entire run:

$$R_{i,|\cdot|}^T = \sum_{t=1}^{T} R_{i,|\cdot|}^t. \tag{5.4}$$

There is one more step involved to normalize this metric if different bet-sizing players use different ranges, as I discuss in the next section.

5.4.1 Comparing different ranges

In Section 5.7 I introduce $RE\text{–}BETS$, an algorithm that adjusts the ranges of bet-sizing players. This leads to a multiplayer betting game where each bet-sizing player may be using a different range. With this in mind, I consider the effect of the bet sizes L_i and H_i on the utilities $u_i^t(L)$ and $u_i^t(H)$. Since we evaluate these utilities keeping the strategies fixed for each player, the utility of a bet is directly proportional to the pot size. The pot size is not, however, directly proportional to the bet size. Given the pot size p, I define the pot-growth function of a bet of size s as

$$G(s) = 1 + 2s. \tag{5.5}$$

If a bet of size s is made and called the new pot size p' is

$$\begin{aligned} p' &= p + 2sp \\ &= p(1 + 2s) \\ &= pG(s). \end{aligned} \tag{5.6}$$

Looking at this equation it's clear that the pot size grows proportional to $G(s)$.

Now let us reconsider Equation 5.3. The signs of $R(L)_i^t$ and $R(H)_i^t$ are related. There are three possibilities: $s_i^t = H_i$, $s_i^t = L_i$, and $L_i < s_i^t < H_i$. The third case can be broken down further, as u_i^t is linear in s_i^t, so one of $R(L)_i^t$ and $R(H)_i^t$ is positive and the other is negative. If $R(H)_i^t$ is positive for the third case we have

$$
\begin{aligned}
|R(L)_i^t| + |R(H)_i^t| &= \pi_{-i}^{\sigma^t}\Big(u_i^t(s_i^t, \sigma^t) - u_i^t(L, \sigma^t)\Big) + \pi_{-i}^{\sigma^t}\Big(u_i^t(H, \sigma^t) - u_i^t(s_i^t, \sigma^t)\Big) \\
&= \pi_{-i}^{\sigma^t}\Big(u_i^t(H, \sigma^t) - u_i^t(L, \sigma^t)\Big) \\
&= \pi_{-i}^{\sigma^t}|u_i^t(H, \sigma^t) - u_i^t(L, \sigma^t)|
\end{aligned} \tag{5.7}
$$

where the final step holds as the right hand side of the equation is guaranteed to be positive. If $R(L)_i^t$ is positive then when $L_i < s_i^t < H_i$ we have

$$
\begin{aligned}
|R(L)_i^t| + |R(H)_i^t| &= \pi_{-i}^{\sigma^t}\Big(u_i^t(L, \sigma^t) - u_i^t(s_i^t, \sigma^t)\Big) + \pi_{-i}^{\sigma^t}\Big(u_i^t(s_i^t, \sigma^t) - u_i^t(H, \sigma^t)\Big) \\
&= \pi_{-i}^{\sigma^t}\Big(u_i^t(L, \sigma^t) - u_i^t(H, \sigma^t)\Big) \\
&= \pi_{-i}^{\sigma^t}|u_i^t(H, \sigma^t) - u_i^t(L, \sigma^t)|
\end{aligned} \tag{5.8}
$$

and again the right hand side must be positive. The cases where $s_i^t = H_i$ or $s_i^t = L_i$ reduce to the same equation:

$$
R_{i,|\cdot|}^t = \pi_{-i}^{\sigma^t}|u_i^t(H, \sigma^t) - u_i^t(L, \sigma^t)|. \tag{5.9}
$$

I now consider the effect that changing the values of L_i and H_i for bet-sizing player i have on this equation. I define the width of the range for player i as

$$
w_i = H_i - L_i. \tag{5.10}
$$

As utility $u_i^t(s, \sigma^t)$ is directly proportional to $G(s)$ I define

$$
u_i^t(s, \sigma^t) = C_i^t G(s) \tag{5.11}
$$

where C_i^t is a constant for player i on iteration t. Applying these two equations and rearranging Equation 5.9 I get

$$
\begin{aligned}
\frac{R_{i,|\cdot|}^t}{\pi_{-i}^{\sigma^t} C_i^t} &= |G(H_i) - G(L_i)| \\
&= |(1 + 2H_i) - (1 + 2L_i)| \\
&= |(1 + 2H_i) - (1 + 2(H_i - w_i)| \\
&= 2w_i.
\end{aligned} \tag{5.12}
$$

I can now write Equation 5.4 as

$$
\begin{aligned}
R_{i,|\cdot|}^T &= \sum_{t=1}^{T} R_{i,|\cdot|}^t \\
&= 2w_i \sum_{t=1}^{T} \pi_{-i}^{\sigma^t} C_i^t.
\end{aligned} \tag{5.13}
$$

With this in mind I define

$$R^t_{i,|\cdot|0} = \frac{R^t_{i,|\cdot|}}{2w_i} \tag{5.14}$$

and

$$R^T_{i,|\cdot|0} = \frac{R^T_{i,|\cdot|}}{2w_i}. \tag{5.15}$$

Now this metric can be used to safely compare bet sizing players using different ranges. In this chapter w_i is constant[4] and so $R^T_{i,|\cdot|0}$ is directly proportional to $R^T_{i,|\cdot|}$ for all bet-sizing players i. In Chapter 6, however, I examine a different way of setting up ranges where the width is no longer constant. Thus for the remainder of the thesis I use Equation 5.15 to measure the relative importance of bet-sizing players.

5.5 Hold'em re-visited

I re-ran the analysis from Section 5.3, this time ranking the bet-sizing players according to $R^T_{i,|\cdot|0}$. The second, fourth and sixth columns of Table 5.2 show the same data as Table 5.1, while the third, fifth and seventh columns of Table 5.2 show the distribution of the 10 bets with the highest $R^T_{i,|\cdot|0}$ values. The important bets tell a slightly different story —- while 8 of these bets are close to $L = 0.8$, two of them moved towards $H = 1$ over the first 8 million iterations, and these two bets had the highest and third highest $R^T_{i,|\cdot|0}$ values.

	$0.80 - 0.82$		$0.82 - 0.98$		$0.98 - 1.0$	
Length	All Bets	Top 10 Bets	All Bets	Top 10 Bets	All Bets	Top 10 Bets
1m	152	8	8	2	0	0
4m	151	8	9	2	0	0
6m	150	8	9	1	1	1
8m	145	8	13	0	2	2
10m	142	8	16	0	2	2
20m	138	8	20	0	2	2
100m	126	9	31	0	3	1
200m	126	9	28	0	6	1

Table 5.2: Bet sizes after different length runs, including bets with high $R^T_{i,|\cdot|0}$ values.

The game value of the equivalent abstract games changed significantly during the first 8 million iterations, while the most important bets were moving. During the final 192 million iterations, however, the game value stayed relatively constant. This result, coupled with the fact that there are important bets at both ends of the $[0.8, 1]$ range, suggested that allowing some of the bets to go lower and others to go higher could result in increased game value. The simplest way to achieve this goal

[4]There are a few minor exceptions to this when using $RE\text{–}BETS$, but only for low-importance bets at the end of some betting sequences.

would be to continue using static ranges, but make them larger. Unfortunately, as I explain in the next section, there are significant disadvantages to using large ranges.

5.6 Tree creation - the small range constraint

When creating the game tree for the multiplayer betting version of a game, there are two issues to be considered:

- What ranges should be used?
- How many bets should I allow in each bet-sequence (what is the depth of the game tree)?

Consider an abstraction of a three round no-limit poker game with initial stacks of 15 big blinds, where only half-pot bets, pot bets and all-in bets are legal for player one, and only pot bets and all-in bets are legal for player two. I will consider all pot sizes in this game in terms of big blinds, and I do not specify the size of the big blind in chips. Instead I will assume it is large enough that no rounding of bet sizes will be required.

As I am only concerned with what happens when bet-sizing players act, I show a reduced version of the game tree, which I call the bet-sizing tree, in Figure 5.1. In making this tree I remove all actions sequences that involve any action other than a bet from the main player (the player to which I will be applying the multiplayer betting transformation). Other betting sequences remain valid and are part of the full game tree, I simply wish to highlight what happens after multiple bet-sizing players have acted.

Each node in Figure 5.1 is labeled by the pot size in big blinds (not in chips) after all bets to this point have been called. For example, the root node is labeled 2 since before any bet can be made, one player must have already put in one big blind and the other player had to have matched it, for a total pot of 2 big blinds. The right child node contains 6 big blinds, since if the main player makes a pot-sized bet (2 big blinds) and is called, the pot would then contain: the current pot (2), plus the bet (2), plus the call (2).

In this game the main player can make three half-pot bets, two pot bets, or two half-pot bets and one pot bet, before an additional bet would require more than the 15 big blinds in the initial stack. For example, to follow the node labeled 16 by a half-pot bet, the main player would have to add 8 big blinds to the pot after having already placed $16/2 = 8$ big blinds into the pot, requiring an initial stack of 16 big blinds. In applying the multiplayer betting transformation to this game, I follow the methodology that I have been using throughout this chapter — I remove all bets from the main player, inserting instead one bet-sizing player at each decision node. I use $L = 0.5$ and $H = 0.5$ for all bet-sizing players. For the opponent player I use the $FCPA$ abstraction, though I do not show opponent bets in the bet-sizing trees.

At this point I must ask the question: does my bet-sizing tree have depth 2 or depth 3?

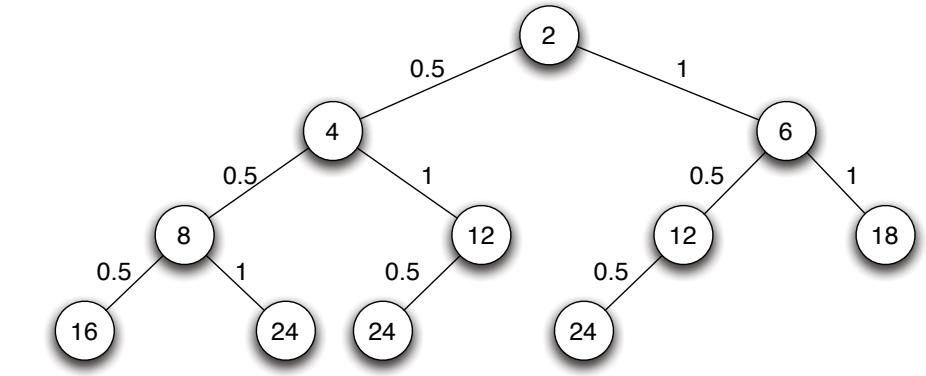


Figure 5.1: Bet-sizing tree for a game with stacks of 15 big blinds, with an abstraction that allows only half pot and pot bets.

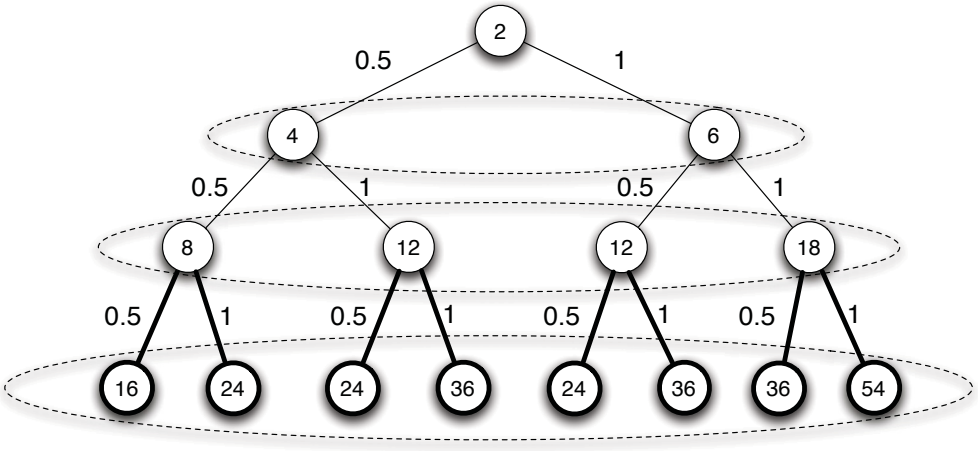


Figure 5.2: Bet-sizing tree for a multiplayer betting transformation of a game with stacks of 15 big blinds, allowing two bets not including the bold actions, or three bets if bold actions are included.

Figure 5.2 shows the bet-sizing tree in the multiplayer betting version of this game, where the bold edges indicate the third time a bet-sizing player acts. The dotted ovals represent information sets, since after a bet-sizing player makes their private action, none of the other players know the resulting pot size. Unfortunately, if I include bet-sequences where three bet-sizing players act in the game tree, then some of the sequences will be invalid. For example, if all bet-sizing players choose a pot bet, after three bet-sizing players have acted there are 54 big blinds in the pot, where each player has contributed 27 big blinds. This is 12 more big blinds than the initial stack size. Alternatively, if I create a game tree where at most two bet-sizing players can act in any bet-sequence, then some legal bets will be missing from the tree. For example, if the first two bet-sizing players to act choose half-pot bets, then it is legal to bet half-pot or pot, so another bet-sizing player could be added to the game tree.

If I do not allow any sequences where three bet-sizing players act, the abstractions I create may not be able to take advantage of a bet action that results in a higher game utility. Unfortunately, the legality of the third bet is dependent on the size of the previous bets and this information is hidden by the information sets, due to the private actions of the bet-sizing players. While missing an opportunity to bet may lower the utility, in order to avoid allowing illegal bets I must assume all bet-sizing players choose H when creating the game tree. This means that if the algorithm selects many small bets, it may not be able to select as many small bets as are possible in the full game. I refer to this disparity between the shortest and longest legal bet-sequence as the "tree-depth problem". It gets worse as the betting ranges get larger, and conversely it can be minimized by selecting small ranges.

Recall that in the previous section we saw the importance of allowing bet sizes to increase or decrease beyond fixed range boundaries. Thus I desire a way to use small ranges, but adjust these ranges as bets become stuck at one of the ends. If I change the ranges, I must reconstruct the game tree, as the new range may permit a different number of bets in the tree. If many of these ranges move down, I need a mechanism for reducing both L and H so that the tree becomes deeper (due to a smaller H) and the preferred bet size can move lower towards the new L. If the algorithm favors a larger bet size, I need to increase both L and H and reconstruct the tree based on a larger H, which may make the tree more shallow.

5.7 Variable range boundaries

Table 5.2 suggested that my expert-informed choice of fixed $[0.8, 1]$ ranges in the Texas hold'em strategy computation was likely suboptimal, since the bet sizes for the most important bets hit the range boundaries. Some bets should be smaller than 0.8 and others should be larger than 1. However, I showed in the previous section that large ranges cause problems due to the large variability of the pot size for a given betting sequence. My approach is to minimize the impact of the tree-depth problem by fixing the size of the range, while moving the two range boundaries.

The first step of this process is creating the initial game tree. I start with a single default range $[L^d, H^d]$. I initialize all ranges to this default range. In the previous section I showed that I should use H^d to determine the depth of the game tree to assure that there are no illegal bets. To do this, the tree is created depth-first, and at each bet-sizing player the H action is taken. This gives me my initial multiplayer betting game G'. This is passed in to my new algorithm, Range Enhanced Bet-size Extraction through Transformation Solving, or RE–$BETS$. This algorithm is outlined in Algorithm 2.

After some initialization of variables, I set $G_1 = G'$ and apply the $BETS$ algorithm. Note, however, that instead of running $BETS$ for many iterations ($T = 200,000,000$) with a single fixed range, I perform R short runs (a few million iterations). At the end of each run r, $BETS$ returns an abstraction A_r. I then change all ranges of G_r so that the new range for each particular bet-sizing

Algorithm 2 The RE–$BETS$ algorithm

Require: Multiplayer betting game G'
Require: R runs
Require: T iterations of $BETS$ per run
Require: L^d
Require: H^d
Require: α
  Set $w = H^d - L^d$
  Set $G_1 = G'$
  for r from 1 to $R - 1$ **do**
    $A_r = BETS(G_r, T)$
    Create new multiplayer betting game G_{r+1} from G_r as follows:
    for each bet-sizing player i in G_r **do**
      Obtain s_i from A_r
      $H_i = s_i + w/2$ and $L_i = s_i - w/2$
    end for
    Set N to the root node of G_{r+1}
    $AdjustTree(G_{r+1}, N, L^d, H^d, \alpha)$
  end for
  $A_R = BETS(G_R, T)$
  return Set of abstractions A

player is centered on the effective bet size in A_r computed for that bet-sizing player by $BETS$. If, therefore, the effective bet size for any bet-sizing player hits a range boundary or was near a range boundary at the end of a run, the bet size will be able to move further in that direction on the next run. I maintain the same width of 0.2 for all ranges in the tree (I revisit this idea in Chapter 6).

After all of the ranges have been changed, there will be some points in the tree where an extra bet will now fit, and at other points bets that were previously legal become illegal. The next step of the algorithm is to add and remove bet sizes for opponent nodes, as well as bet-sizing players. Algorithm 3, AdjustTree, performs this task. Each time a bet-sizing player is encountered I must check that the L and H actions are both legal, and if they are not I must adjust them accordingly. This is done by Algorithm 4, AdjustRange. Note that it is not necessary to call AdjustRange on newly added bet-sizing players, provided that I set $L^d \geq 0.5$, as 0.5 is always at least a minimum bet. I return to this discussion in Section 6.5.

AdjustTree (Algorithm 3) recursively performs a depth-first traversal of the tree, starting from the root node. As the legality of different bet sizes does not depend on the chance nodes, I only need to consider each betting sequence once, independent of the cards. For this reason I skip over chance nodes. At each opponent node I cycle through all the bets in the opponent abstraction. If the bet is not already in the tree, but is now legal, it is added to the tree, and a node below it is added with actions fold, call and all-in. No other actions are added at the new node at this point; all other legal betting actions will be added when that node is reached by AdjustTree. If, conversely, the bet being considered was in the tree, but is now illegal, it and the sub-tree below it are removed. A similar process is performed for the main player nodes, using H^d to determine if a bet-sizing player can be

Algorithm 3 AdjustTree

Require: G
Require: N
Require: L^d
Require: H^d
Require: α
  if N is an opponent node **then**
    for each bet size b in the opponent abstraction **do**
      if b is not the in tree **then**
        if b is legal **then**
          Add bet b to G at N, and a node below b with fold, call and all-in
        end if
      else
        if b is no longer legal **then**
          remove b and the sub-tree below from G at N
        end if
      end if
    end for
  else if N is not a chance node **then**
    Denote the size of an all-in bet at this decision node as s_{all-in}
    if there is no bet-sizing player at N **then**
      if $H^d < s_{all-in} - \alpha$ **then**
        Add a bet-sizing player to G at N using $[L^d, H^d]$, and an opponent node below this with fold, call and all-in
      end if
    else
      Denote this bet-sizing player i
      Let L_i, H_i be the L and H values used by bet-sizing player i at N
      $(L_i, H_i, useBet) = AdjustRange(L_i, H_i, \alpha, s_{all-in})$
      if $useBet == false$ **then**
        Remove bet-sizing player i and the sub-tree below from G
      end if
    end if
  end if
  if N is a chance node **then**
    Sample c from chance
    Take action c
    Denote the current node N'
    $G = AdjustTree(G, N', L^d, H^d, \alpha)$
  else
    for each action a at N **do**
      Take action a
      if a leads to bet-sizing player i **then**
        Take action H_i
      end if
      Denote the current node N'
      $G = AdjustTree(G, N', L^d, H^d, \alpha)$
    end for
  end if
  return G

Algorithm 4 AdjustRange

Require: L
Require: H
Require: α
Require: s_{all-in}
  $useBet = true$
  $w = H - L$
  if $L < (s_{minbet} + \alpha)$ **then**
    $L = s_{minbet} + \alpha$
    $H = L + w$
    if $H > s_{all-in} - \alpha$ **then**
      $H = s_{all-in} - \alpha$
      if $H - L < w/2$ **then**
        $useBet = false$
      end if
    end if
  else if $H > s_{all-in} - \alpha$ **then**
    $H = s_{all-in} - \alpha$
    $L = H - w$
    if $L < (s_{minbet} + \alpha)$ **then**
      $L = s_{minbet} + \alpha$
      if $H - L < w/2$ **then**
        $useBet = false$
      end if
    end if
  end if
  return $L, H, useBet$

added to the tree. If there is a bet-sizing player at this node (whether it was already there or has just been added), AdjustRange is now called.

In AdjustRange (Algorithm 4) I first check if, for the bet-sizing player i, L_i has decreased to the point where it is within a small constant α of the minimum bet size s_{minbet}. If so, I increase L_i to $L_i = s_{minbet} + \alpha$ and adjust H_i upwards to maintain the constant range size w.[5] Conversely, if bet-sizing players that acted previous to player i have increased their H values, then H_i may be larger than an all-in bet size. This may also happen after I adjust the range upwards to accommodate the minimum bet size. If this happens, I set $H_i = s_{all-in} - \alpha$ and set L_i to accommodate the fixed range size. If both constraints cannot be met while maintaining a fixed width w, I reduce the range width as necessary. If, however, the range size decreases to less than half of the original width w, I remove bet-sizing player i from the game, along with the sub-tree beneath that player. At this point the minimum bet is quite close to an all-in bet, and so having a bet-sizing player here is unlikely to add value.

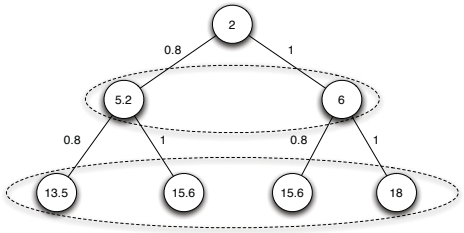
Once AdjustTree finishes, I have an abstraction A_r. The entire process is repeated R times, producing a set A containing R abstractions.

Consider applying the $RE\text{–}BETS$ algorithm to the game from Section 5.6, but instead using the default range of $[0.8, 1]$. I start with the bet-sizing tree shown in Figure 5.3(a) (in these examples I round the pot size to the nearest 0.1 big blinds for ease of presentation, $RE\text{–}BETS$ uses exact values). Note that this tree has a structure identical to Figure 5.2 if the bold edges are not included. For the purposes of this example, I will assume that the $BETS$ algorithm always picks the L action for all bet-sizing players. In reality $BETS$ may (and most likely will) produce different bet sizes for each bet-sizing player. Thus the first run of $BETS$ yields bet sizes of 0.8 for all bet-sizing players. The new ranges for all bet-sizing players are $[0.7, 0.9]$. The new bet-sizing tree is shown in Figure 5.3(b). While the pot sizes are now different, the structure of this tree is the same as that of Figure 5.3(a), as after two 0.9 bets a bet of size $H^d = 1$ would still be illegal.

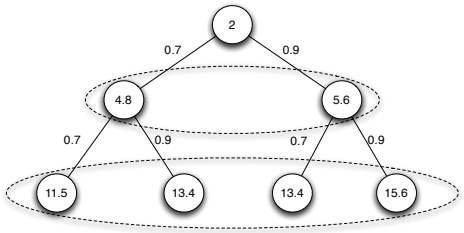
On the next run of $BETS$ I again assume all L actions are picked, and thus this leads to bet sizes of 0.7. This produces the bet-sizing tree shown in Figure 5.3(c). Similarly run $r = 3$ produces Figure 5.3(d). Run $r = 4$, however, produces a different structure, shown in Figure 5.3(e). With ranges of $[0.4, 0.6]$, the pot size is *at most* 9.7 big blinds after two bet-sizing players have acted. This means after a bet of size $H^d = 1$ the maximum possible pot size is 29 big blinds, which is allowed as each player starts with 15 big blinds.

Note that $RE\text{–}BETS$ also adds opponent bets to the game tree when they become legal. In this example the opponent abstraction is $FCPA$, so the opponent's bet size (pot) is the same as the default bet size for new ranges (H^d). This means that on run $r = 4$ after $BETS$ completes, for any action sequence in which two bet-sizing players have acted, a pot sized bet will be legal. $RE\text{–}BETS$ will add this action to the game tree for the opponent in these spots.

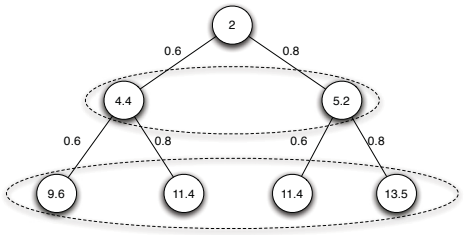
[5] A value of $\alpha = 0.01$ was used in all of my experiments



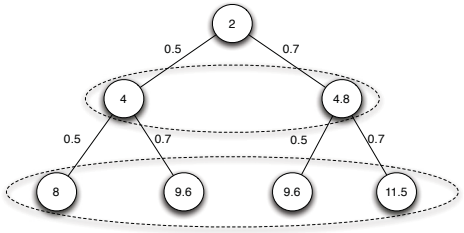
(a) Bet-sizing tree for multiplayer betting game G_1



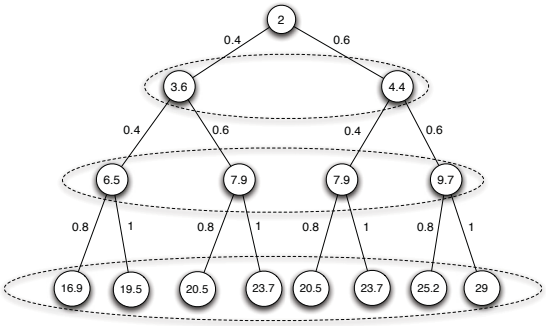
(b) Bet-sizing tree for multiplayer betting game G_2



(c) Bet-sizing tree for multiplayer betting game G_3



(d) Bet-sizing tree for multiplayer betting game G_4



(e) Bet-sizing tree for multiplayer betting game G_5

Figure 5.3: Bet-sizing trees of multiplayer betting games produced by $RE\text{–}BETS$. The values at each node represent the number of big blinds in the pot at that point, rounded to the nearest 0.1.

5.8 Leduc and Texas hold'em revisited

I applied $RE\text{–}BETS$ to the restricted Leduc game I introduced in Section 5.2, using default ranges of $[0.8, 1]$ for all bet-sizing players. Using 2, 4, 6, 8 and 10 million iterations per run of $BETS$, the game value of the equivalent abstract games converged each time within 10 iterations of $RE\text{–}BETS$, beating the value of the $FCPA$ abstraction by 21% and 9% for players one and two respectively, as compared to the 12% and 7.7% gains I showed in Section 5.2. The game value of the abstractions produced by $RE\text{–}BETS$ levelled off in each case before the 10 iterations had completed. This shows that $RE\text{–}BETS$ can out perform a single long run of $BETS$, even when the width of the ranges used for that single run (0.7) is larger than the width (0.2) used in $RE\text{–}BETS$. The equivalent abstract games contained many important bet sizes outside the range $[0.8, 1.5]$ — for the 10 million iteration run of player one, 3 of the top 4 bets, as ranked by $R^T_{i,|\cdot|0}$, were greater than 1.5.

This Leduc game has a cap of two bets per round and starting stacks of 400 antes. With a two bet per round cap and large stacks, I was able to use large ranges in Section 5.2 and apply $BETS$ while avoiding the tree-depth problem described in Section 5.6. The betting sequence length is at most 4 bets and the starting stack is large enough to make 4 bets of 1.5 pot each. With $RE\text{–}BETS$, this restriction is no longer necessary. So I applied $RE\text{–}BETS$ to an unrestricted Leduc game, with 200 big blind stacks. Using the same ranges and run lengths for $BETS$, and $FCPA$ as the opponent abstraction, the abstractions I created improved over a baseline $FCPA$ abstraction by $\approx 30\%$ and $\approx 43\%$ for player one and player two respectively.

5.8.1 Texas hold'em

In the Texas hold'em experiment described by Table 5.2 most bets were close to an edge of their range after 1 million iterations, with the 10 top bets, as ranked by $R^T_{i,|\cdot|0}$, being close to an edge within 8 million iterations. With this result in mind I applied $RE\text{–}BETS$ with various values of T (the number of iterations $BETS$ will execute) between 2 and 10 million, and $R = 20$ (R is the number of times $BETS$ is run). I found that for both players, the game value of the equivalent abstract games I created would initially increase after each run of $BETS$, eventually peaking. While $T = 2,000,000$ was not enough, using $T = 8,000,000$ led to good performance, being slightly edged out by $T = 10,000,000$. Using $T = 10,000,000$ the game value of the equivalent abstract games peaked after $R = 8$ for player one and $R = 6$ for player two, improving over the baseline $FCPA$ action abstraction by $\approx 48\%$ and $\approx 88\%$ respectively. I call these peak-value abstractions $P1_8$ and $P2_6$ and I use these for the analysis presented in the remainder of this chapter.

Good no-limit agents typically use at least two bet sizes in their action abstractions - usually pot and half pot. Since half pot bets are small, they increase tree depth, so the number of half pot bets is usually restricted [25, 58]. I created a number of action abstractions that use unrestricted pot bets, plus one half pot bet, once per round on specific rounds. I also created a larger action abstraction

including 0.5, 0.75, 1, 3 and 11 pot bets, allowing only one 0.5 bet and one 0.75 bets on each of the flop, turn and river. This is the action abstraction used by the winner of the 2011 no-limit bankroll instant run-off division of the ACPC. These abstractions were made asymmetrically - one player was given extra betting options while the other used the opponent betting abstraction ($FCPA$). These abstractions and my generated abstractions are listed in Table 5.3, along with the number of (I, a) (information set, action) pairs they contain. The number of extra (I, a) pairs does not depend on which player has the added bets. The amount of memory required to apply the CFR algorithm is proportional to the number of (I, a) pairs in the game [68].

Name	Pot fractions in abstraction	# (I, a) pairs
$FCPA$	1	1,781,890
H_{PF}	1, 0.5 once on pre-flop	3,867,090
H_F	1, 0.5 once on flop	3,820,140
H_T	1, 0.5 once on turn	3,646,890
H_R	1, 0.5 once on river	3,143,140
H_{FTR}	1, 0.5 once on flop, turn and river	9,115,390
$ACPC_{2011}$	1, 3, 11, 0.5 and 0.75 once on flop, turn and river	31,135,210
$P1_8$	Bet-sizing agent, player 1, 8 runs	2,930,050
$P2_6$	Bet-sizing agent, player 2, 6 runs	3,121,775

Table 5.3: Number of (I, a) pairs in various action abstractions, each with $FCPA$ opponents, in 5 bucket perfect recall card abstractions of Texas hold'em.

As all of these action abstractions share the same opponent, this allows me to directly compare each of their game values. I used chance-sampling CFR to find ϵ-Nash equilibria, and then I computed best responses to these equilibria, obtaining upper and lower bounds on the game value for each player in each case. Figures 5.4 and 5.5 shows the results for some of the abstractions that were used for players one and two, respectively.[6] The Y-axis represents milli-big blinds won by player two (the dealer), thus player one wishes to reduce this value, while player two wishes to increase it. The bounds are very tight - for the basic $FCPA$ abstraction a run of about 200 million iterations would in practice be considered long enough to produce an acceptable ϵ for this size of game. To obtain these bounds I ran chance-sampled CFR between 2 and 4 billion iterations on each of the bet-sizing abstractions and 5 billion iterations on each of the other abstractions. Not all of the abstractions listed in Table 5.3 are included in these plots. H_{FTR} overlapped bounds with $ACPC_{2011}$ for both players, with $ACPC_{2011}$ being slightly better in both cases. For clarity H_{FTR} is not included in the plots. For player one (Figure 5.4) H_{PF} showed little improvement over $FCPA$, and H_F, H_T and H_R all performed roughly the same. Thus only $FCPA$, H_F and $ACPC_{2011}$ are shown. For player two (Figure 5.5), abstractions H_T and H_R are not included in the plot as they were only slightly better than $FCPA$.

[6] See the appendix for the exact values of the curves and their bounds for all game value curves presented in this dissertation

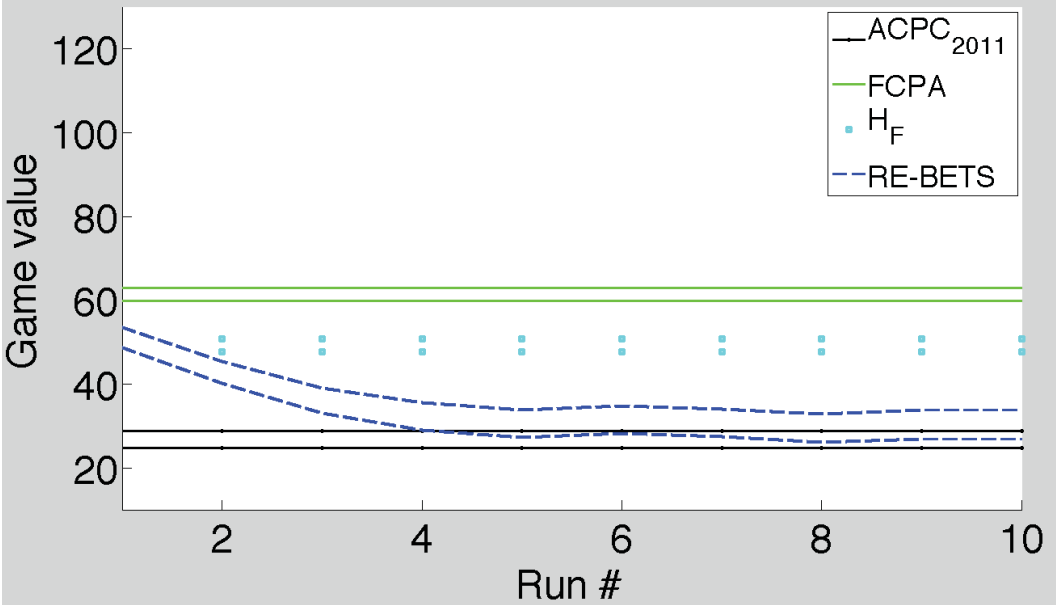


Figure 5.4: Bounds on game values of various betting abstractions for player one against an $FCPA$ player two, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to reduce this number.

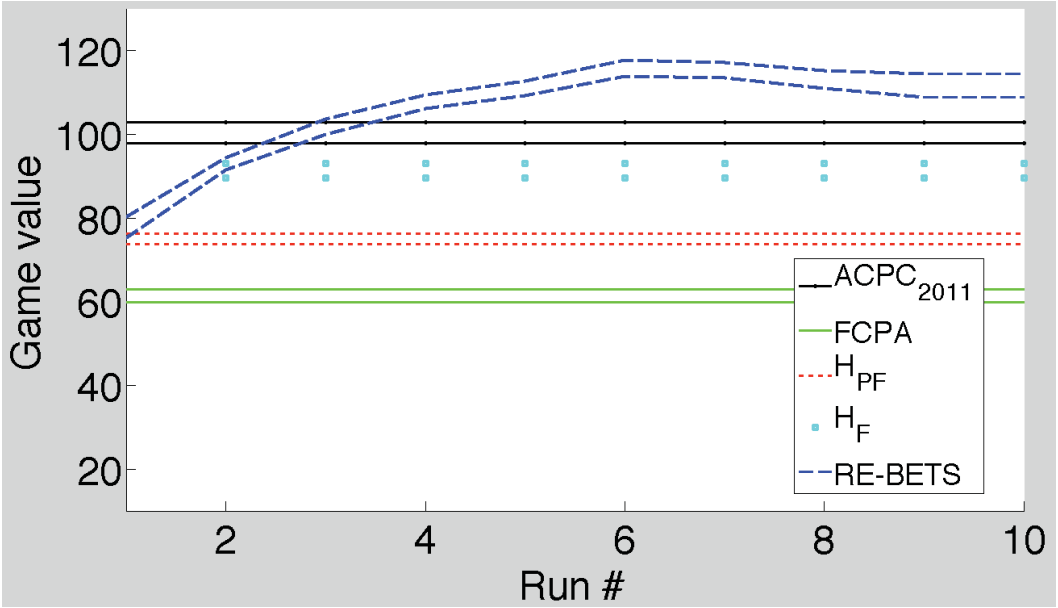


Figure 5.5: Bounds on game values of various betting abstractions for player two against an $FCPA$ player one, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number

As shown in Figure 5.5, abstraction $P2_6$ out performs all other abstractions, beating $ACPC_{2011}$ by $\approx 15\%$ after 6 iterations of $RE\text{–}BETS$. For player one, abstraction $P1_8$ had bounds on its peak

game value close to those of $ACPC_{2011}$ and H_{FTR}, and easily beat the rest of the abstractions, as shown in Figure 5.4. Additionally, we can see from Table 5.3 that abstractions $P1_8$ and $P2_6$ are about one tenth the size of $ACPC_{2011}$. Smaller betting abstractions are always favoured over larger ones with similar value, as this allows for refinement of the card abstraction. In Chapter 7 I examine how the value of the action abstractions I create changes as the card abstraction is refined.

A histogram showing the distribution of the top 10% of bets for abstractions $P1_8$ and $P2_6$, ranked according to $R^T_{i,|\cdot|_0}$, is shown in Figure 5.6. Each bet size was rounded to the nearest 0.1 — the 0.7 bar includes bets from $0.650 - 0.749$. We can see that no one bet size is preferred - a variety of bets from 0.2 to 1.5 are used. The important bet sizes are different for each player - $P1_8$'s range from 0.2 to 0.7 along with a couple of large bets > 1, while $P2_6$'s vary from 0.4 to 1.

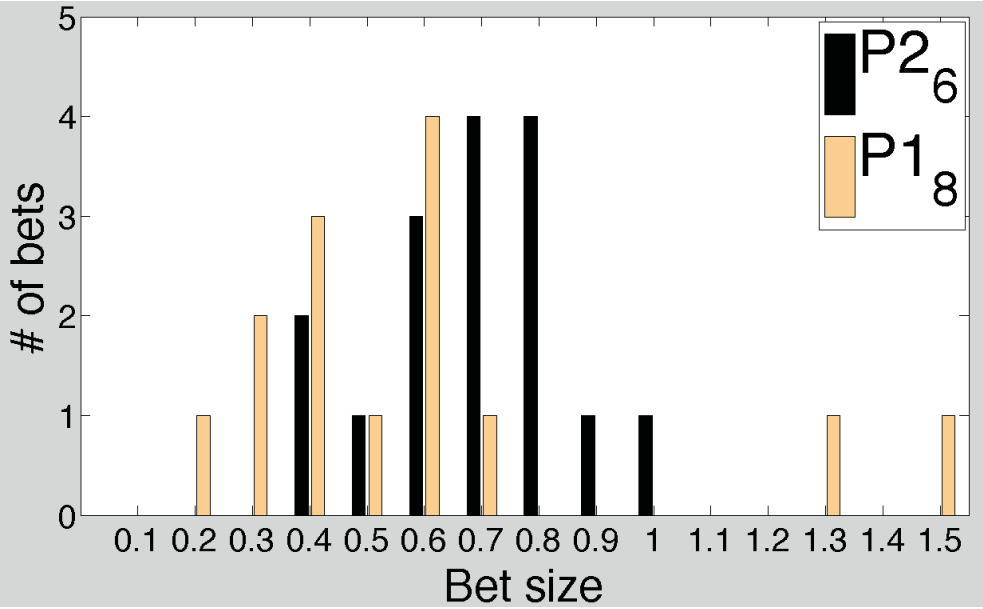


Figure 5.6: Distribution of important bet sizes for abstractions $P1_8$ and $P2_6$. Bets are rounded to the nearest 0.1

5.9 ACPC 2012 competition entry

The $RE\text{–}BETS$ algorithm was used in the creation of a no-limit Texas hold'em poker agent that was entered in the 2012 ACPC. In creating the action abstractions for each player I used a different opponent action abstraction and a different card abstraction from the rest of this chapter. The action abstractions were generated using a small card abstraction, then combined with a much larger card abstraction and passed in to a state-of-the-art distributed implementation of the Monte Carlo CFR (MCCFR) algorithm which created the final production agent [18, 40, 46]. I defer the discussion of the choices of card and opponent action abstraction and the merits of using a small card abstraction to create the action abstractions and a big card abstraction to create the final agents to Chapters 6

and 7. In this section I simply present the results of this application.

The Texas hold'em poker agent I created won the no-limit bankroll instant run-off division and placed second in the no-limit bankroll division. It used the $ACPC_{2011}$ abstraction introduced in Section 5.8.1 as the opponent action abstraction and a default range of $[0.5, 0.7]$. A value of $T = 10,000,000$ was used for each run of $BETS$. It uses an imperfect recall card abstraction, generated using the k-means clustering state abstraction technique of Johanson et al. mentioned in Section 3.7.1 [42]. It has 169 buckets on the pre-flop,[7] 100 buckets on each of the flop and turn, 50 buckets on the river for player one and 100 buckets on the river for player two. Note that, as this is an imperfect recall card abstraction, it is smaller than the perfect recall 5 bucket abstraction used in the rest of this chapter. In a perfect recall card abstraction the number of buckets is exponential in the rounds — a perfect recall 5 bucket game has $5^4 = 625$ bucket sequences that must be considered on the river. Initially, I used 100 river buckets in this imperfect recall card abstraction for both players. For player one the first few iterations of $RE\text{–}BETS$ yielded very similar results with just 50 river buckets, and so that smaller abstraction was used to speed convergence.

In Chapter 7 I discuss in detail the question of how to evaluate an imperfect recall card abstraction. In creating this agent, however, I used a simple technique: $RE\text{–}BETS$ was run until the top bets, as ranked by $R^T_{i,|\cdot|0}$, were no longer pinned at the edges of their ranges after the $BETS$ algorithm had finished running. In Section 5.8.1 we saw that for a 5 bucket card abstraction and an $FCPA$ opponent action abstraction, it only took $R = 6$ and $R = 8$ runs of $BETS$ for players one and two respectively for the game value curves to peak. In this case, however, the top bets, as ranked by $R^T_{i,|\cdot|0}$, did not level off as fast. In total 24 runs of $BETS$ were used for player 1 and 9 runs of $BETS$ were used for player 2.

While a very small card abstraction was used by $RE\text{–}BETS$ to obtain my action abstraction, the next step in the creation of the final agent was to apply this action abstraction to a game with a larger card abstraction. The action abstractions I generate are compatible with any card abstraction, as the bet-sizing players choose their bets based solely on the betting sequence, independent of the cards. The size of card abstraction that can be used is directly related to the size of my action abstractions — the smaller the action abstraction, the more room there is for refinement of the card abstraction. To get an idea of the size of the action abstractions created by $RE\text{–}BETS$ I compare them to a baseline action abstraction using $ACPC_{2011}$ for both players.

The winning entry of the bankroll instant run-off division of the ACPC in 2011 used this $ACPC_{2011}$ action abstraction along with an imperfect recall card abstraction with 169 pre-flop buckets and 3700 buckets on the flop, turn, and river. That agent has $2,685,582,334$ (I, a) pairs, requiring 40 gigabytes of RAM to use a CFR-based solution technique. In contrast, if the action abstractions I generated are paired with the same card abstraction the resulting game has $787,297,938$ and $1,235,465,864$ (I, a) pairs, requiring 12 and 18.5 gigabytes of RAM respectively for players one

[7] There are only 169 possible unique hands in the pre-flop of Texas hold'em.

and two. So the card abstraction was increased to have 169 pre-flop buckets and 18630 buckets on the flop, turn and river. This combination of my action abstraction and a large card abstraction was still small enough to require only 42 gigabytes of RAM for player 2 and 24 gigabytes of RAM for player 1 when loaded by the MCCFR algorithm [18, 40, 46]. MCCFR was run for approximately 18 days on a cluster using nodes of 4 AMD 6172 processors with 12 cores each, running at $2.1GHz$. All 48 cores were used in the calculation. In total 141 billion iterations were run for player one, and 93 billion iterations for player two. Each of these nodes has 256 gigabytes of RAM, and so a much larger card abstraction could have, theoretically, been solved. The larger the card abstraction, the longer it takes to solve the game — this 18630 bucket abstraction was used due only to time constraints.

As with any production agent, this agent had to have a translation algorithm to account for bets made by opponents that were not contained within the $ACPC_{2011}$ opponent abstraction. I used the translation system designed by Schnizlein et al. [59, 58], discussed in Section 3.8.2. In addition to this there is one other component to this agent — a technique known as thresholding is used, where we alter the strategy of the agent such that any actions it would choose less than $t\%$ of the time are never chosen, with this probability being distributed across the rest of the actions with greater than $t\%$ probability of being chosen. While removing some of the theoretical guarantees of an ϵ-Nash equilibrium, this technique has been shown to perform better in practice in matches against other poker agents [17].

The results of the bankroll instant run-off division of the 2012 ACPC competition are shown in Figure 5.7, courtesy of the ACPC website [1]. This agent was entered on behalf of the University of Alberta Computer Poker Research Group (CPRG), who use the name "hyperborean" for their entries in these competitions. Each column of the table represents an elimination round of the run-off competition, after which the weakest agent is removed from the group and the scores are recalculated. The values given in each cell of this table are the totals for the agents listed in the rows against the all agents that remain in the corresponding round of the competition. As you can see in rounds 0 and 1 hyperborean was in third place, then in rounds 2 and 3 hyperborean rose to second overall. During rounds 0 through 3 little rock led by a significant margin, due to the fact that it beat the worst agents by a larger margin. However from round 4 on hyperborean was the clear leader. Hyperborean beat every other agent in the individual one on one matches.

While this agent performs well, it is worth noting that a false assumption was made in creating it — that $T = 10,000,000$ would be sufficient to adjust ranges in the appropriate way when using this card abstraction and opponent action abstraction. Better action abstractions can be generated if larger values of T are used. In addition, this reduces the value of R needed for convergence. I will discuss this further in Chapter 7.

	Round 0	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7	Round 8	Round 9
hyperborean	586 ± 30	527 ± 28	530 ± 31	273 ± 23	223 ± 25	167 ± 27	167 ± 31	181 ± 37	174 ± 24	161 ± 36
tartanian5	597 ± 27	536 ± 27	509 ± 30	159 ± 30	97 ± 27	-7 ± 26	17 ± 29	4 ± 34	-76 ± 27	-161 ± 36
neo.poker.lab	534 ± 23	456 ± 23	424 ± 25	198 ± 23	86 ± 26	16 ± 28	-58 ± 31	-78 ± 32	-97 ± 23	–
little.rock	1116 ± 27	929 ± 28	907 ± 30	298 ± 26	128 ± 25	37 ± 27	-24 ± 35	-107 ± 35	–	–
sartre	112 ± 25	19 ± 25	-21 ± 29	56 ± 22	-57 ± 22	-97 ± 23	-101 ± 27	–	–	–
hugh	483 ± 18	459 ± 19	422 ± 22	71 ± 18	-49 ± 17	-117 ± 18	–	–	–	–
spewy.louie	355 ± 28	257 ± 29	204 ± 33	-230 ± 31	-427 ± 31	–	–	–	–	–
lucky7.12	-809 ± 57	-719 ± 47	-1009 ± 52	-826 ± 45	–	–	–	–	–	–
azure.sky	-196 ± 65	-1209 ± 60	-1966 ± 68	–	–	–	–	–	–	–
dcubot	-1097 ± 15	-1254 ± 16	–	–	–	–	–	–	–	–
uni.mb.poker	-1681 ± 47	–	–	–	–	–	–	–	–	–

Figure 5.7: Results of the 2012 ACPC no-limit bankroll instant run-off competition, in milli-big blinds.

Chapter 6

Multiple fixed-ratio ranges

In this chapter I consider generalizing $RE\text{–}BETS$ to use more than one bet-sizing agent at each decision node, as well as ranges set by fixing the ratio of the rate at which the high and low bets cause the pot size to grow. In Section 6.1 I outline my methodology for choosing the number of iterations of $BETS$ to use each time it is run. Section 6.2 explores the effect that different default ranges have on the results of $RE\text{–}BETS$. This leads me to the idea of fixed-ratio ranges, detailed in Section 6.3, which have many advantages over fixed-width ranges. I introduce the $RED\text{–}BETS$ algorithm, which uses fixed-ratio ranges, in Section 6.4. Next I explore using more than one bet-sizing player at each decision node. I implement this in the $REDM\text{–}BETS$ algorithm in Section 6.5, which leads to an improvement in game value over $RED\text{–}BETS$. The analysis of Section 6.6, however, shows that we can maintain this gain while only using multiple bet-sizing players at a few key points in the game tree. The combination of $REDM\text{–}BETS$ and this idea culminate in my final algorithm, $REFINED\text{–}BETS$. Section 6.7 shows the application of $REFINED\text{–}BETS$ to Texas hold'em, and Section 6.8 shows that $REFINED\text{–}BETS$ helps avoid the local maxima problem introduced in Section 4.5.

I continue to use the same 5 bucket perfect recall card abstraction of no-limit Texas hold'em in this chapter as was used in Chapter 5, as well as an opponent action abstraction of $FCPA$. I consider imperfect recall card abstractions and more complex opponent action abstractions in Chapter 7.

6.1 Choosing T for $BETS$

In Section 5.8 I applied $RE\text{–}BETS$ (Algorithm 2) with a default range of $[0.8, 1]$ to a 5 bucket perfect recall abstraction of no-limit Texas hold'em. I used the movement of the most important bet-sizing players, as ranked by $R^T_{i,|\cdot|0}$, to decide how many iterations I should use when running $BETS$, settling on $T = 10,000,000$. For player one this led to equivalent abstract games that increased in game value until the peak point, maintaining the peak value after each subsequent run of $BETS$. For player two, however, the game value peaked after 6 runs, then dipped slightly before levelling off. In order to see if the dip in the curve is related to the number of iterations per run of

$BETS$, I repeated the experiment from Section 5.8 using $T = 20,000,000$. This led to a very minor improvement in game value at the peak point. Notably, however, the peak value was maintained for both players even after running $BETS$ 20 times.

Now I consider why $T = 10,000,000$ is enough to reach approximately the same peak value reached when using $T = 20,000,000$, but not enough to maintain that peak (for player two). It is important to realize that even once the game value has peaked, the game tree continues to change each time $BETS$ is run. The tree size can grow if, for example, bet-sizing players with relatively minor influence on the game value choose smaller and smaller bet sizes. This leads to extra (inconsequential) bet-sizing players being added to the tree during the AdjustTree phase (Algorithm 3, Section 5.7) of RE–$BETS$. If the game tree grows, larger values of T are required to achieve the same game value. In general, for the poker games studied in this dissertation, the size of the abstractions generated using RE–$BETS$ (and its variants I introduce in this chapter) increases each time $BETS$ is run.

With this in mind, the first time I run $BETS$ I use a larger value of T than I expect I will need, much as was done in Sections 5.3 and 5.5. I then use the movement of the most important bet-sizing players over the course of this run, as ranked by $R^T_{i,|\cdot|0}$, as an indication of what value of T will be needed for subsequent runs to convergence. For clarity of presentation, I do not delve in to the details of how I choose T in each case. As I discussed in Section 5.5, the value of the abstraction generated on the first run of $BETS$ changes very little once important bets are close to the edge of a range, so using more iterations on the first run of $BETS$ has little effect on the game value curves.

6.2 Changing the default range

In Section 5.8 I used a default range of $[0.8, 1]$ for all bet-sizing players. Recall that the default range is used to initialize the ranges of the bet-sizing players, both in the initial game tree and during AdjustTree (Algorithm 3), when it is used to decide whether to add a bet-sizing player to the tree (see Section 5.7). When considering the choice of a default range, the following questions come to mind:

- How does the choice of default range affect the number of times $BETS$ must be run before the game value curve peaks?
- Does the choice of default range affect the value at the peak point?

To address these questions, I ran the experiment from Section 5.8 three times, using a different default range for all bet-sizing players each time. For the reasons described in Section 6.1 I used $T = 20,000,000$ for each run of $BETS$. The results of the first 10 runs, for player one, are shown in Figure 6.1.

The first thing to note looking at this figure is that all three default ranges reach approximately the same game value after the tenth run of $BETS$. Additionally, examining the top 10% of bets

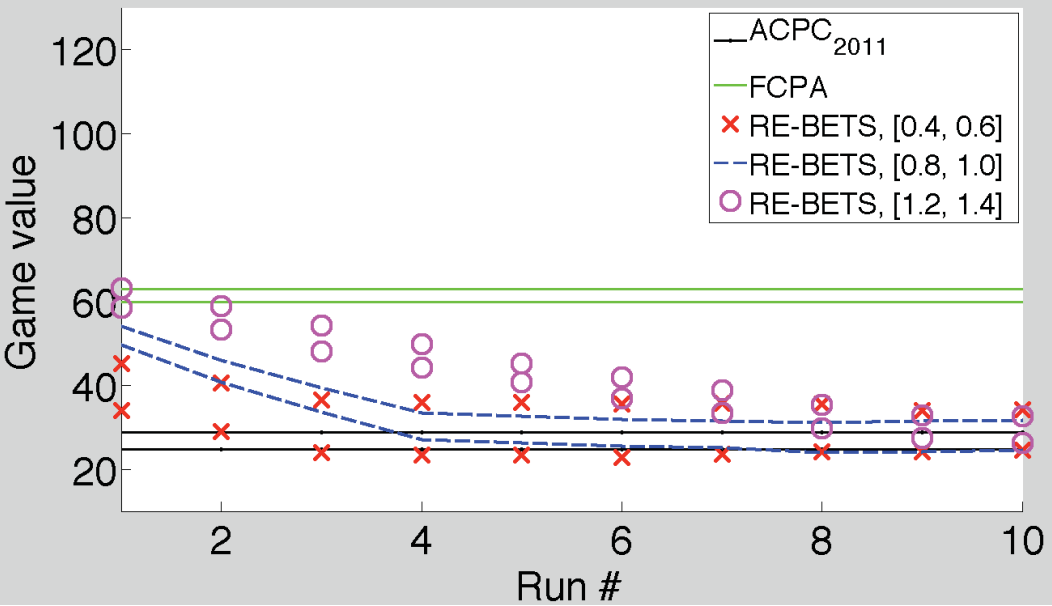


Figure 6.1: Bounds on game values of abstractions produced by $RE\text{–}BETS$, using three different default ranges, for player one against an $FCPA$ player two, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to reduce this number.

for these final abstractions as ranked by $R^T_{i,|\cdot|0}$, they were all within 0.1 of their value in $P1_8$, the abstraction I created for player one in Section 5.8.1 in each case. This suggests that when we allow a maximum of one bet-sizing player at every decision node, different starting conditions for $RE\text{–}BETS$ may lead to very similar action abstractions with approximately equivalent game values, given the right value of T in each case. Even if there are multiple different-valued maxima of the multiplayer betting game, as discussed in Section 4.5, $RE\text{–}BETS$ is converging on maxima with very similar values.

The same experiment for player two is shown in Figure 6.2. These results are very similar to that of player one, except for the fact that the smallest default range, $[0.4, 0.6]$, peaks at a slightly lesser game value than the other two. This result is similar to the problem discussed in Section 6.1 — because the default range is centred on a smaller bet size, bets are being added to the game tree more often. They are added as soon as a bet of size $H^d = 0.6$ becomes legal. This leads to bet sequences which have very little influence on the game value tending to be longer, and thus the size of the equivalent abstract games is bigger. So while $T = 20,000,000$ is enough to converge on approximately the same game value when the default range is centred on a larger bet size, when it is centred on a smaller bet size we need more iterations of $BETS$ to achieve the same game value.

Something that separates the results of $RE\text{–}BETS$ using different default ranges, for both players, is the number of short runs it takes to converge on the game value. The reason for this becomes clear when we consider that in each case the bet sizes are converging on the same values, indepen-

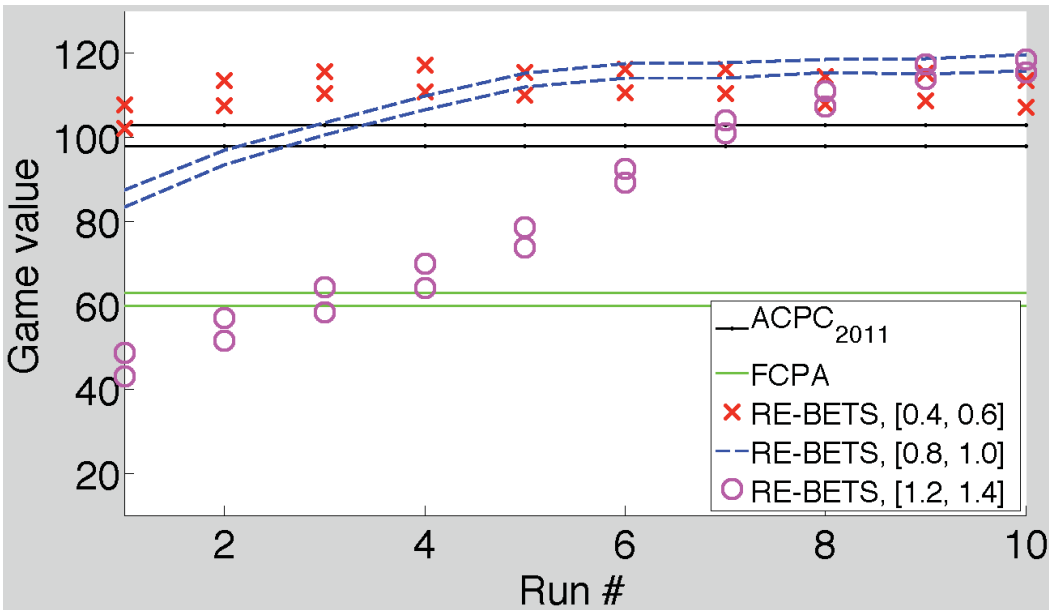


Figure 6.2: Bounds on game values of abstractions produced by $RE\text{–}BETS$, using three different default ranges, for player two against an $FCPA$ player one, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number.

dent of where they start. The most that a range can move in one short run is 0.1, which would happen if the average effective bet size is one of the edges (so either $B(\overline{P(H)}_i^T) = H$ or $B(\overline{P(H)}_i^T) = L$). As the majority of the important bets are < 1 (see Figure 5.6), the larger the bet size at the centre of the default range is, the longer it will take to get to these bet sizes. While for this game the simplest solution is to choose a default range centred on a small bet size, such as $[0.4, 0.6]$, this leads to larger games that require larger values of T. Additionally, changing the card abstraction and opponent action abstraction may lead to a very different distribution of bets, so it is not clear that a range centred on a smaller bet size will always be better (see Chapter 7). In the next section I describe a variable-width alternative to these fixed-width ranges that, among other advantages, causes the width of ranges to increase when the centre of the range moves towards larger bet sizes, and decrease when it moves towards smaller bet sizes.

6.3 Fixed-ratio ranges

In all experiments to this point I have used a fixed range width of 0.2. Now I consider how the choice of L or H made by a bet-sizing player using fixed-width ranges of 0.2 affects the growth of the pot. To do this recall that in Section 5.4 I introduced the pot-growth function $G(s)$ of bet size s, restated here:

$$G(s) = 1 + 2s. \tag{6.1}$$

This is defined such that if the pot size is p then after a bet of size s the new pot size p' is

$$p' = pG(s). \tag{6.2}$$

I now define the pot-growth ratio P_i of a bet-sizing player i as the ratio of the pot size after the H action to the pot size after the L action:

$$\begin{aligned} P_i &= \frac{pG(H_i)}{pG(L_i)} \\ &= \frac{G(H_i)}{G(L_i)}. \end{aligned} \tag{6.3}$$

The smallest default range from the experiments in Section 6.2 is $[0.4, 0.6]$. Applying Equation 6.2, the pot size p' becomes

$$p' = p(1 + 2(0.4)) = 1.8p \tag{6.4}$$

after the L action and

$$p' = p(1 + 2(0.6)) = 2.2p \tag{6.5}$$

after the H action, with a pot-growth ratio of $2.2/1.8 = 1.22$. Applying the same calculation to the largest default range used in Section 6.2, $[1.2, 1.4]$, produces pot sizes of $3.4p$ and $3.8p$ for L and H actions, respectively, and a pot-growth ratio of $3.8/3.4 = 1.12$. As you can see in this example, as a fixed-width range move towards larger bets the pot-growth ratio goes down. If, however, I set ranges such that $P_i = c$ for all bet-sizing players i, where c is a constant, then as the range moves up the width of the range will increase, and as the range moves down it will decrease. I call such ranges "fixed-ratio ranges". Next I examine how setting ranges in this way affects the tree-depth problem from Section 5.6.

6.3.1 The range-size constraint revisited

In Section 5.6 I showed that to avoid making illegal bets, the game tree of a multiplayer betting game must be created with the assumption that all bet-sizing players will choose the H action. If throughout an action sequence all bet-sizing players choose action L, at the end of that action sequence it's possible that there will be enough chips left for an extra bet-sizing player using the default range $[L^d, H^d]$, but this player will not exist in the game tree. I called this the "tree-depth problem". Using small fixed-width ranges helps to reduce this problem, which is part of the motivation for $RE\text{–}BETS$. Here I will show that I can improve upon this approach by using specially chosen fixed-ratio ranges. This approach allows me to make certain guarantees with regard to how bad this problem can get.

Note that the tree-depth problem was introduced in Section 5.6 using a game without all-in bets. The existence of all-in bets helps to reduce the impact of this problem, and I assume I am working with a game with all-in bets throughout this section, for convenience. A similar analysis applies if there is no all-in bet in the game.

To calculate the bet size s that would put a player all-in on their n^{th} bet, I get the n^{th} root of S, the number of big blinds in the stacks. I then use Equation 6.1 to solve for s. For example, let's assume $S = 200$, as it does in most of my experiments. The fifth root of 200 is $200^{1/5} \approx 2.8853$, meaning that the pot grows by a factor of $G(s) \approx 2.8854$ after each bet s. Solving this for s gives $s \approx 0.943$. Thus if five bets of size $s = 0.943$ are made, then the last bet will be approximately an all-in bet (within rounding error). Similarly, to determine how many bets are required to get all-in after six bets of equal size, I calculate $200^{1/6} \approx 2.4183$, leading to a bet size of $s \approx 0.709$. As before, if six bets of size $s = 0.709$ are made, the last bet will be approximately an all-in bet.

Before continuing, I consider the effect of bets made by an opponent using an opponent betting abstraction (making fixed bet sizes). Any such bet will reduce the number of chips remaining in the stacks, while the ratio of the largest and smallest possible pot sizes does not change, as the opponent's bet size is known by all players. Thus, opponent bets reduce the impact of the tree-depth problem, as reducing the stack sizes means that fewer bet-sizing players can act in this action sequence. With this in mind I start by considering only the worst-case scenario: action sequences in which the opponent does not make any bets. In such action sequences all bets are made by bet-sizing players.

Now I re-visit the tree-depth problem in the initial game tree that will be created by RE–$BETS$ if we use the default range $[0.71, 0.94]$ for all of our bet-sizing players in this game with $S = 200$. Note that I have rounded H^d down from the fifth root, and L^d up from the sixth root. Using RE–$BETS$, the game tree is created by substituting a bet of size H^d for all bet-sizing players (see Section 5.7). As $H^d = 0.94$ is slightly less than 0.943, at most five bet-sizing players can act in a single action sequence. If each of these bet-sizing players chooses the H action, then there will remain only a very small all-in bet. If, instead, each of these five bet-sizing players chooses the L action, there will be enough chips left for an all-in bet of slightly less than $L = 0.71$, as six $s = 0.709$ bets put a player all-in on the sixth bet. What this means is that even if I changed RE–$BETS$ to create the game tree assuming all bet-sizing players chose L, I would create the exact same tree! Thus using a default range of $[0.71, 0.94]$ has eliminated the tree-depth problem on the initial run of $BETS$.

Now consider the following theorem.

Theorem 1. *In any multiplayer betting game with stacks of S big blinds, if x is the n^{th} root of S, y is the $(n+1)^{th}$ root of S and for every bet-sizing player i $P_i = x/y$, then after n bet-sizing players have acted the ratio of the largest to the smallest possible pot size is $S^{1/(n+1)} = y$.*

Proof. After one bet-sizing player acts the ratio between the possible pot sizes will be x/y, by the definition of P_i. The largest ratio of possible pot sizes after n bets is $(x/y)^n$. By the definitions of x and y this becomes

$$\frac{x^n}{y^n} = \frac{S}{y^n} = \frac{S}{y^{n+1}/y} = \frac{S}{S/y} = y. \tag{6.6}$$

□

If $RE\text{–}BETS$ is modified to keep the pot growth ratio P_i constant for every bet sizing player i, then even as ranges move up and down the ratio of the largest to smallest possible pot size is constant for a given number of actions n by bet-sizing players. Theorem 1 guarantees that if I set the default range $[L^d, H^d]$ using any two roots of S, in the worst case the smaller pot size will reach the larger pot size with a bet of size L^d, since this grows the pot by $G(L^d) = S^{1/(n+1)}$.

Additionally, recall that in $RE\text{–}BETS$ ranges are never increased to the point where H_i is an all-in for any bet-sizing player i. Instead H_i is always kept slightly smaller than an all-in bet. Bet-sizing player i is only removed from the game tree if the range becomes so small that it is impossible to keep L_i greater than a minimum bet, H_i less than an all-in bet, and maintain a width of at least half of $H^d - L^d$ (this is done by AdjustRange, Algorithm 4). With this in mind I adjust H^d such that $G(H^d)$ is *slightly less* than the n^{th} root of S, as was done in the example at the beginning of this section. Doing this creates an all-in bet after n bet-sizing players act, which will then remain unless the n^{th} bet-sizing player to act is removed from the game tree (at which point there will still be an all-in bet after the previous bet-sizing player acts). This all-in bet may be of little utility if all H actions are used, however it provides an additional betting option of size up to slightly less than L^d after a sequence of L actions. This guarantees the elimination of the tree-depth problem on the first run of $BETS$, for arbitrary choice of n. As I did earlier in the section, I use two decimal points for my default ranges, rounding H^d up and L^d down.

I must clarify my statement that the tree-depth problem is eliminated. I defined the tree-depth problem as a situation in which I create a different tree if I assume all H bets are made than if I assume all L bets are made. This is conceptually simple, and makes sense in the initial multiplayer betting game. However, in the general case where each bet-sizing player is using different ranges, it is helpful to consider a different metric. Instead of determining simply whether or not a bet of size L^d would be legal after a sequence of L actions, consider the size of an all-in bet after a sequence of L actions (even in a game with no all-in bet, simply consider how large an all-in bet would be). Before the first run of $BETS$, the size of this bet is slightly less than L^d, which is reasonable. The size of this all-in bet will change, however, as bet-sizing players move their ranges up and down. As bet-sizing players move their ranges towards smaller bets, this all-in bet gets larger, until an additional bet-sizing player is added. As they move their ranges towards larger bets, this all-in bet gets smaller, unless a bet-sizing player is removed. Cases where a bet-sizing player gets added do reduce the size of the final all-in, but increase the ratio of greatest to smallest pot size, and thus make it possible for an even bigger all-in if bet-sizing players continue to move their ranges towards smaller bets. Cases where a bet-sizing player is removed aren't a concern, as we don't remove them unless their H bet is very small, meaning the size of the all-in is also quite small at this point.

With fixed-ratio ranges, the size of this all-in bet size only depends on the action sequence with the most bet-sizing players in it, the fixed ratio P, and the default bet size used to add a new bet-

sizing player. With fixed-width ranges, it also matters if the bet-sizing players acting have ranges centred on very small bet sizes or very big ones. Ranges centred on very small bet sizes make the problem worse, as they increase the ratio of possible pot sizes. Ranges centred on large bet sizes reduce the problem, as they decrease this ratio. Consider, however, that ranges centred on small bet sizes are the problematic ones, as they may increase the number of possible bet-sizing players to act in a single sequence, and conversely ranges centred on large bet sizes are less of a concern. So using fixed-width ranges makes the problem worse when it is already bad, and helps alleviate the problem when it is getting better naturally.

6.3.2 Choice of n

There are two practical considerations that must be made when choosing a value of n for multiplayer betting games. The first of these is whether we are fixing one player's bets and applying the transformation to the other player, or applying the transformation to both players. In the former case, as mentioned in the previous section, any bet made by the fixed-betting player reduces the problem, as that player's bet size is known by all players. The second consideration is the number of rounds in the game. If there are four rounds (as is common in many poker games), then there can only be four bets made by one player if the other player does not bet or raise. Thus, if one player uses fixed bets, then for there to be five actions by bet-sizing players in a given betting sequence, then there must be at least one fixed bet made in that sequence.

I consider choosing n for no-limit Texas hold'em with an (arbitrary) fixed betting opponent (in the next chapter I will consider an opponent abstraction of $ACPC_{2011}$). One option is to set the value of n very low. This will, however, mean that the default range will be centred on a very large bet size, and the pot growth ratio P will be quite large. For example, a choice of $n = 3$ corresponds to a range of $[1.39, 2.42]$ and a pot growth ratio of $P = 1.54$. With this default range, it will take many runs to reach small bet sizes, so if we expect that many bet-sizing players will tend to choose small bet sizes (as was the case in Section 5.7, illustrated in Figure 5.6), this range is not ideal. Additionally, a large pot growth ratio will make the tree-depth problem worse if many bet-sizing players choose smaller bets.

The opposite approach would be to use a large value of n, such as $n = 7$. This would lead to a default range of $[0.47, 0.56]$ and a pot growth ratio of $P = 1.1$. The first problem encountered here is that $L^d = 0.47$ will not always be legal — when facing a bet of 11 pot (which is included in the $ACPC_{2011}$ abstraction) the minimum bet (raise) size is $s_{minbet} = 0.48$. Also, for games with only four rounds and a fixed betting opponent, this is overly pessimistic — any betting sequence with seven bet-sizing players acting in it must also contain three opponent bets. These three opponent bets significantly reduce the impact of the tree-depth problem.

One last consideration comes from poker theory. Once enough chips have been place in the pot, both players become "pot committed". It is generally accepted among poker theorists that being

pot committed means that it would almost always be a mistake for either player to fold, and that if either player bets they should usually make an all-in bet [9, 10, 13, 28, 51, 62]. Miller et al. argue that once your remaining stack contains less chips than the pot, you are pot committed [13]. Thus once an all-in bet is $s_{all-in} < 1$, adding a bet-sizing player at this point seems unlikely to improve the game value of the abstraction. Putting all of this together, I choose a value of $n = 5$ for my experiments in no-limit Texas hold'em for the remainder of this dissertation. This leads to a default range of $[0.71, 0.94]$, and a pot growth ratio of $P = 1.19$.

6.4 RED–$BETS$ algorithm

To use fixed-ratio ranges instead of fixed-width ranges, I must alter RE–$BETS$ accordingly. I define the default pot-growth ratio P^d as

$$P^d = \frac{1 + 2H^d}{1 + 2L^d}. \tag{6.7}$$

For each bet-sizing player i, I must be able to take a bet size s_i and determine new range boundaries H_i and L_i. To do this I rewrite Equation 6.7 as

$$P^d = \frac{1 + 2(s_i + w_i/2)}{1 + 2(s_i - w_i/2)} \tag{6.8}$$

where

$$H_i = s_i + w_i/2 \tag{6.9}$$

and

$$L_i = s_i - w_i/2. \tag{6.10}$$

In Equation 6.8 P^d and s_i are known, and I must solve for w_i. I do this as follows:

$$\begin{aligned} P^d &= \frac{1 + 2s_i + w_i}{1 + 2s_i - w_i} \\ P^d(1 + 2s_i - w_i) &= 1 + 2s_i + w_i \\ P^d + 2s_iP^d - w_iP^d &= 1 + 2s_i + w_i \\ P^d + 2s_iP^d - 1 - 2s_i &= w_i(1 + P^d). \end{aligned} \tag{6.11}$$

Simplifying I get

$$\begin{aligned} w_i &= \frac{P^d(1 + 2s_i) - (1 + 2s_i)}{P^d + 1} \\ w_i &= \frac{(1 + 2s_i)(P^d - 1)}{P^d + 1}. \end{aligned} \tag{6.12}$$

I now introduce an update to the RE–$BETS$ algorithm, which I call Range Enhanced and Depth-fixed Bet size Extraction through Transformation Solving (RED–$BETS$). It is very similar

Algorithm 5 The RED–$BETS$ algorithm

Require: Multiplayer betting game G'
Require: R runs
Require: T iterations of $BETS$ per run
Require: L^d
Require: H^d
Require: α
  Set $P^d = \frac{H^d}{L^d}$
  Set $G_1 = G'$
  for r from 1 to $R - 1$ **do**
    $A_r = BETS(G_r, T)$
    Create new multiplayer betting game G_{r+1} from G_r as follows:
    for each bet-sizing player i in G_r **do**
      Obtain s_i from A_r
      // The line below is the significant change from RE–$BETS$ (Algorithm 2, Section 5.7)
      $w_i = \frac{(1+2s_i)(P^d-1)}{P^d+1}$
      $H_i = s_i + w_i/2$ and $L_i = s_i - w_i/2$
    end for
    Set N to the root node of G_{r+1}
    $AdjustTree(G_{r+1}, N, L^d, H^d, \alpha)$
  end for
  $A_R = BETS(G_R, T)$
  return Set of abstractions A

to RE–$BETS$, with the difference being that instead of fixed-width ranges I use fixed-ratio ranges. I do this by updating the width using Equation 6.12. RED–$BETS$ is detailed in Algorithm 5.

Note that RED–$BETS$ continues to use AdjustTree (Algorithm 3) and AdjustRange (Algorithm 4, Section 5.7). It is worth noting that an adjustment could be made to AdjustRange to maintain fixed-ratio ranges in the event that $L_i < s_{minbet} + \alpha$ or $H_i >$ all-in - α. This adds an extra level of complexity that is very unlikely to add any value to the algorithm. To understand why, consider that there are four possible scenarios that can be encountered in AdjustRange:

1. L_i is too small so it is moved up, and H_i is moved up by the same amount
2. H_i is too large so it is moved down, and L_i is moved down by the same amount
3. The range must be squeezed to accommodate both of these constraints.
4. Accommodating these constraints would squeeze the range to less than half its original width, so we remove this bet-sizing player

If the first scenario occurs, the range is moved upwards. If AdjustRange was modified to maintain fixed-ratio ranges, H_i would move up by more than the L_i (see Section 6.3.1). This, in turn, makes it more likely that a bet-sizing player acting after player i will have an H action that will now be larger than all-in. So in this case leaving AdjustRange unchanged is a conservative move, possibly making some ranges smaller than necessary while avoiding some extra changes throughout the tree. In the second scenario the opposite effect occurs — because the range is moving down,

L_i must be reduced by less than H_i if I wish to maintain fixed-ratio ranges. By leaving this as is, however, I do not cause any tree-depth problems for other bet-sizing players, because i is the last bet-sizing player to act in the action sequence.

The third and fourth scenarios would not be affected by an attempt to maintain fixed-ratio ranges, as the ranges are reduced in size from both ends. Also, it should be noted that any changes made to ranges by AdjustRange are temporary — every time $BETS$ runs I create new ranges centred on the effective bet sizes, using Equation 6.12 to create new, fixed-ratio ranges. If, for example, the second scenario is encountered and a range is moved down, then even if the next time $BETS$ runs it chooses the exact centre of the bet-sizing player in question's current range, the new range will be set by Equation 6.12 and will thus move the L bet size up accordingly.

For these reasons I decided to not to change AdjustRange and so I continue to use the original version (Algorithm 4).

There is an additional advantage of RED–$BETS$ on top of those listed in Section 6.3.1. Provided the default range starts low enough, the worst-case number of runs of $BETS$ required to reach a bet size decreases. I have chosen a default low bet that starts at $L = 0.71$ (see Section 6.3.2), and considering that, when facing a pot bet, a min bet is $s_{minbet} \approx 0.33$, there is not far to go to reach very small bets. The scenario that I have to be concerned about is if the range is headed towards a large bet, say over 2 pot. This will take a minimum of one run of $BETS$ per 0.1 change in bet size when using RE–$BETS$. As the default range here starts at width 2.3 ($[0.71, 0.94]$), and increases as the centre of the range moves towards larger bet sizes, the number of runs of $BETS$ required to reach large bets decreases.

As an example, consider the minimum number of runs it would take for each of RE–$BETS$ and RED–$BETS$ to reach a bet size of $s = 2.5$. Using RE–$BETS$ with a starting range of $[0.8, 1]$ it would take at least 15 runs — one run per 0.1 change in bet size. Using RED–$BETS$ with a starting range of $[0.71, 0.94]$, it would take a minimum of 10 runs.

6.4.1 Application of RED–$BETS$ to 5 bucket Texas hold'em

I ran RED–$BETS$ with a default range of $[0.71, 0.94]$, using an $FCPA$ opponent action abstraction. I set $T = 20,000,000$ for each run of $BETS$. Figures 6.3 and 6.4 show the resulting game value curves for players one and two, respectively, compared to the curves for the same game using RE–$BETS$ with $T = 20,000,000$ iterations for each run of $BETS$ and a default range of $[0.8, 1]$. As you can see, both range types hit approximately the same game value, while the fixed-ratio ranges converge slightly faster in both cases. I use fixed-ratio ranges in my experiments for the remainder of the dissertation.

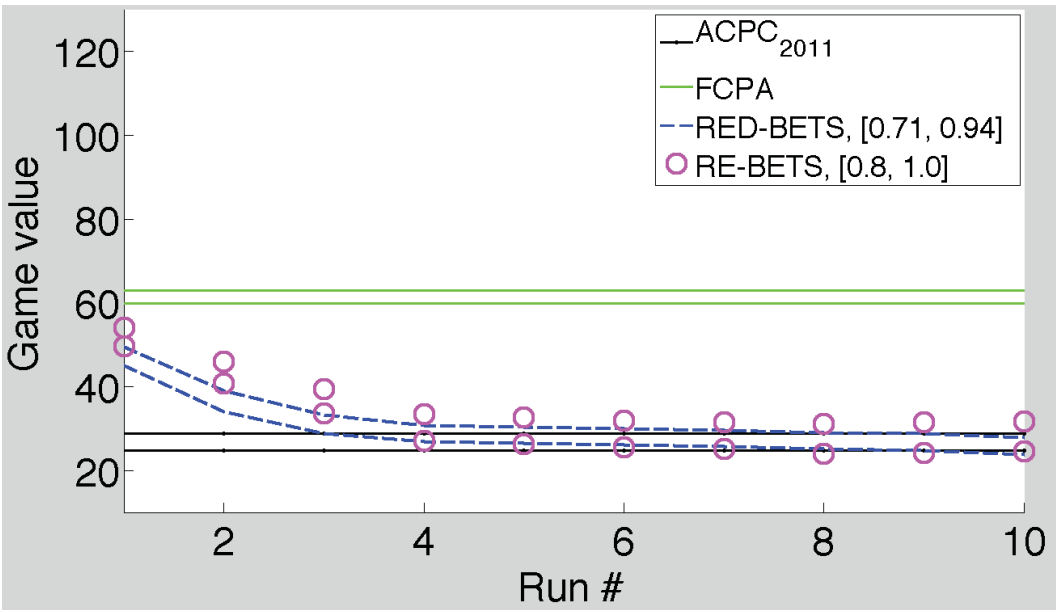


Figure 6.3: Bounds on game values of abstractions using fixed-ratio vs fixed-width ranges for player one against an $FCPA$ player two, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to decrease this number.

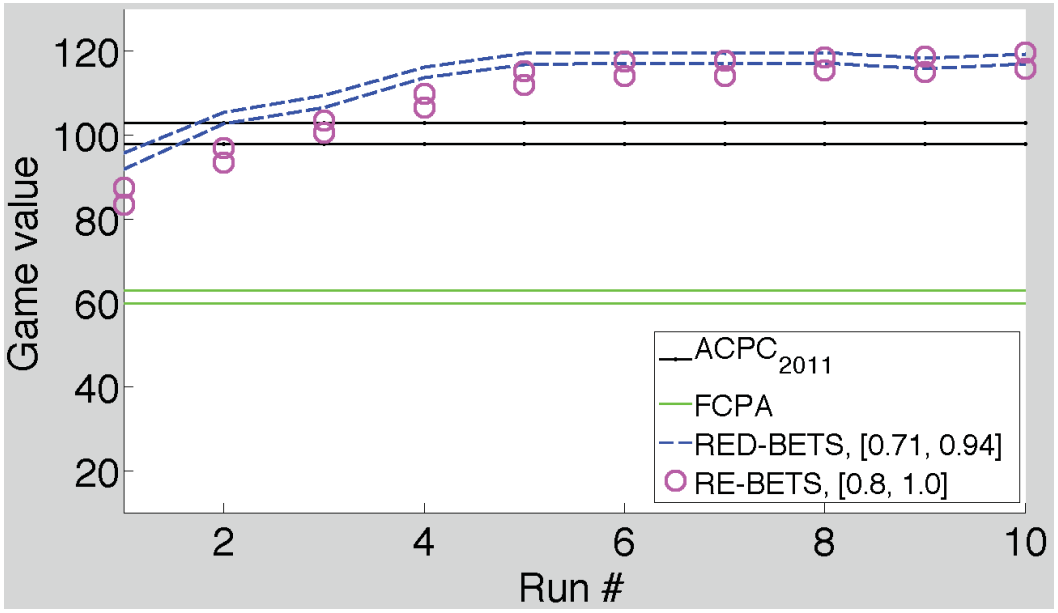


Figure 6.4: Bounds on game values of abstractions using fixed-ratio vs fixed-width ranges for player two against an $FCPA$ player one, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number.

6.5 Multiple ranges

All of the results I have presented so far have used only one bet-sizing player at each decision node. A natural question to ask is whether or not using multiple bet-sizing players at a single decision node would result in a significant gain in game value. Having multiple bet-sizing players at the same decision node allows for strategic behaviour that is not possible when there is just one bet-sizing player. For example, it may be advantageous to mix between large and small bet sizes, and do so with different percentages based on different private cards. For example, if I have a very good hand, when I bet I may wish to pick a large bet size 80% of the time and a smaller bet size 20% of the time. If I have a mid-range hand, I may adjust these percentages to 50% each. By mixing between bets in different ways with different private cards I am, on average, revealing some information about what cards I hold. The value gained by getting more money in the pot when my hand is better, however, may outweigh that disadvantage. This type of mixing between different sized bets based on private information is a common recommendation in the poker literature [10, 27, 51, 62].

Before considering how to expand RED–$BETS$ to allow for multiple bet-sizing players at a single decision node, I discuss how I go about choosing multiple default ranges. I denote the i^{th} default range $[L_i^d, H_i^d]$. For the smallest default range I use $L_1^d = 0.5$. As this is the smallest bet size that is always legal in no-limit hold'em, any time I add a bet-sizing player using $[L_1^d, H_1^d]$, $L_1^d = 0.5$ it is guaranteed to be legal, independent of the opponent abstraction that is used. Using $P = 1.19$ and substituting in to Equation 6.3 this leads to an H_1^d value of

$$\begin{aligned} \frac{1 + 2H_1^d}{1 + 2(0.5)} &= 1.19 \\ H_1^d &= 0.69. \end{aligned} \tag{6.13}$$

Thus the smallest default range is $[0.5, 0.69]$. For the second default range I choose H_2^d such the pot growth ratio of an H_2^d sized bet would be slightly smaller than the fourth root of 200. This leaves an all-in bet in the game tree after any action sequence in which four bet-sizing players using $[L_2^d, H_2^d]$ have acted. For a discussion of why it is advantageous to choose $G(H^d)$ to be slightly less than a root of the stack size in big blinds (200) see Section 6.3.1. As $200^{1/4} \approx 3.7606$, I set $H_2^d = \frac{3.76-1}{2} = 1.38$. Again using $P = 1.19$ and Equation 6.3 this gives me $L_2^d = 1.08$. I do some experiments where I allow two bet-sizing players throughout the tree, and in these experiments I use default ranges of $[0.5, 0.69]$ and $[1.08, 1.38]$.

In some additional experiments I add a third default range along with these two. H_3^d is picked, again, to have a pot growth ratio slightly smaller than a root of 200, this time the third root. The third root of 200 is $200^{1/3} \approx 5.848$, so I set $H_3^d = \frac{5.84-1}{2} = 2.42$. Once more setting $P = 1.19$ and using Equation 6.3 I get $L_3^d = 1.95$. A summary of the default ranges used for these multiple range experiments, along with the significance of each choice of default range, is given in Table 6.1.

Maximum bet-sizing players per node	Default ranges	Significance
1	$[0.71, 0.94]$	After 5 0.94 bets there is a small all-in
2	$[0.5, 0.69]$	0.5 is the smallest bet that is always legal
	$[1.08, 1.38]$	After 4 1.38 bets there is a small all-in
3	$[0.5, 0.69]$	0.5 is the smallest bet that is always legal
	$[1.08, 1.38]$	After 4 1.38 bets there is a small all-in
	$[1.95, 2.42]$	After 3 2.42 bets there is a small all-in

Table 6.1: Default ranges used in multi-range experiments

6.5.1 $REDM$–$BETS$

I expand RED–$BETS$ to multiplayer betting games with multiple bet-sizing players at each decision node. I call this new algorithm Range Enhanced and Depth-fixed Multiple Bet size Extraction through Transformation Solving ($REDM$–$BETS$). Initially, I form the game tree by adding all default ranges i whenever H_i^d is legal. After each run of $BETS$ I follow the same approach as RED–$BETS$, but cycling over all ranges. I continue to use fixed-ratio ranges with the ratio of $P = 1.19$ calculated using $n = 5$ for all the reasons described in Section 6.3.2. Note that with $numRanges$ default ranges there can be many places in the tree with less than $numRanges$ bet-sizing players, as there may not be enough chips left for all H_i^d bets at these points. The full algorithm is given by Algorithm 6.

Algorithm 6 The $REDM$–$BETS$ algorithm

Require: Multiplayer betting game G'
Require: R runs
Require: T iterations of $BETS$ per run
Require: $numRanges$, the number of default ranges to use
Require: $L^d[]$, an array of size $numRanges$
Require: $H^d[]$, an array of size $numRanges$
Require: α
  Set $P^d = \frac{H^d[1]}{L^d[1]}$
  Set $G_1 = G'$
  for r from 1 to $R - 1$ **do**
    $A_r = BETS(G_r, T)$
    Create new multiplayer betting game G_{r+1} from G_r as follows:
    for each bet-sizing player i in G_r **do**
      Obtain s_i from A_r
      $w_i = \frac{(1+2s_i)(P^d-1)}{P^d+1}$
      $H_i = s_i + w_i/2$ and $L_i = s_i - w_i/2$
    end for
    Set N to the root node of G_{r+1}
    // The line below is the significant change from RED–$BETS$ (Algorithm 5, Section 6.4)
    $AdjustTreeMultiBets(G_{r+1}, N, numRanges, L^d[], H^d[], \alpha)$
  end for
  $A_R = BETS(G_R, T)$
  return Set of abstractions A

This algorithm differs only slightly from RED–$BETS$. It now takes as input $numRanges$,

the number of default ranges to use, and arrays of $L^d[]$ and $H^d[]$ of size $numRanges$. The game tree now has numbered branches for each bet-sizing player — if a bet-sizing player is added using default range j, then from that point on it is "using range j". This way I can tell if I need to try adding a bet-sizing player using default range j, or if that player has already been added in the past (but may be using a very different range now).

The call to AdjustTree has been replaced by a new algorithm, AdjustTreeMultiBets (Algorithm 7). AdjustTreeMultiBets is, conceptually, a simple extension of AdjustTree (Algorithm 3, Section 5.7) to multiple ranges. Where AdjustTree assumed only one range was used, AdjustTreeMultiBets loops over all ranges j, either attempting to add a bet-sizing player using default range j or calling AdjustRange on the bet-sizing player already using range j. If only one range is passed to AdjustTreeMultiBets, it becomes equivalent to AdjustTree.

AdjustTreeMultiBets uses 1-indexed arrays; that is, $L^d[]$ and $H^d[]$ go from 1 to $numRanges$. I use 1-indexed arrays in algorithms for the remainder of the dissertation.

6.5.2 Application of $REDM$–$BETS$ to 5 bucket Texas hold'em

For one default range I used $T = 20,000,000$ as I did in Sections 6.4 and 6.1. Again I used an opponent action abstraction of $FCPA$. Following the methodology outlined in Section 6.1, for two default ranges I used $T = 45,000,000$ and for three default ranges I used $T = 70,000,000$. In all three cases I completed 10 runs of $BETS$. The game value curves for player one and player two are shown in Figures 6.5 and 6.6, respectively.

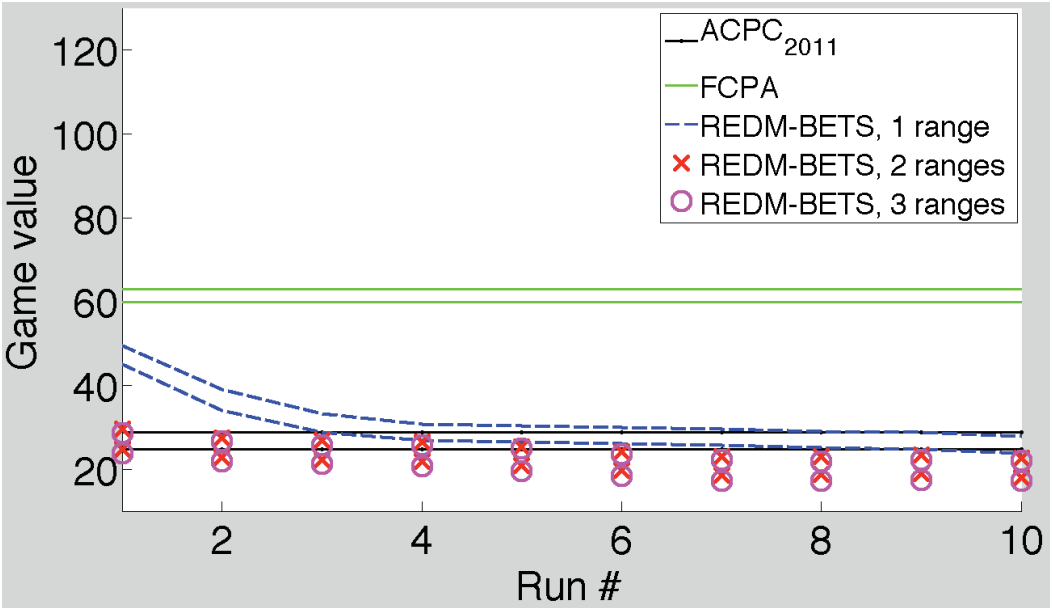


Figure 6.5: Bounds on game values of abstractions using one, two and three default ranges for player one against an $FCPA$ player two, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to decrease this number.

Algorithm 7 AdjustTreeMultiBets

Require: G, N, α
Require: $numRanges$, the number of default ranges to use
Require: $L^d[]$, an array of size $numRanges$
Require: $H^d[]$, an array of size $numRanges$
  if N is an opponent node **then**
    for each bet size b in the opponent abstraction **do**
      if b is not the in tree **then**
        if b is legal **then**
          Add bet b to G at N, and a node below b with fold, call and all-in
        end if
      else
        if b is no longer legal **then**
          remove b and the sub-tree below from G at N
        end if
      end if
    end for
  else if N is not a chance node **then**
    Denote the size of an all-in bet at this decision node as s_{all-in}
    // The addition of this loop is the significant change from $AdjustTree$
    // (Algorithm 3, Section 5.7)
    for j from 1 to $numRanges$ **do**
      if there is no bet-sizing player using range j at N **then**
        if $H^d[j] < s_{all-in} - \alpha$ **then**
          Add a bet-sizing player to G at N using $[L^d[j], H^d[j]]$, and an opponent node below this with fold, call and all-in
        end if
      else
        Denote this bet-sizing player i
        Let L_i, H_i be the L and H values used by bet-sizing player i at N
        $(L_i, H_i, useBet) = AdjustRange(L_i, H_i, \alpha, s_{all-in})$
        if $useBet == false$ **then**
          Remove bet-sizing player i and the sub-tree below from G
        end if
      end if
    end for
  end if
  if N is a chance node **then**
    Sample c from chance
    Take action c
    Denote the current node N'
    $G = AdjustTree(G, N', L^d, H^d, \alpha)$
  else
    for each action a at N **do**
      Take action a
      if a leads to bet-sizing player i **then**
        Take action H_i
      end if
      Denote the current node N'
      $G = AdjustTreeMultiBets(G, N', numRanges, L^d[], H^d[], \alpha)$
    end for
  end if
  return G

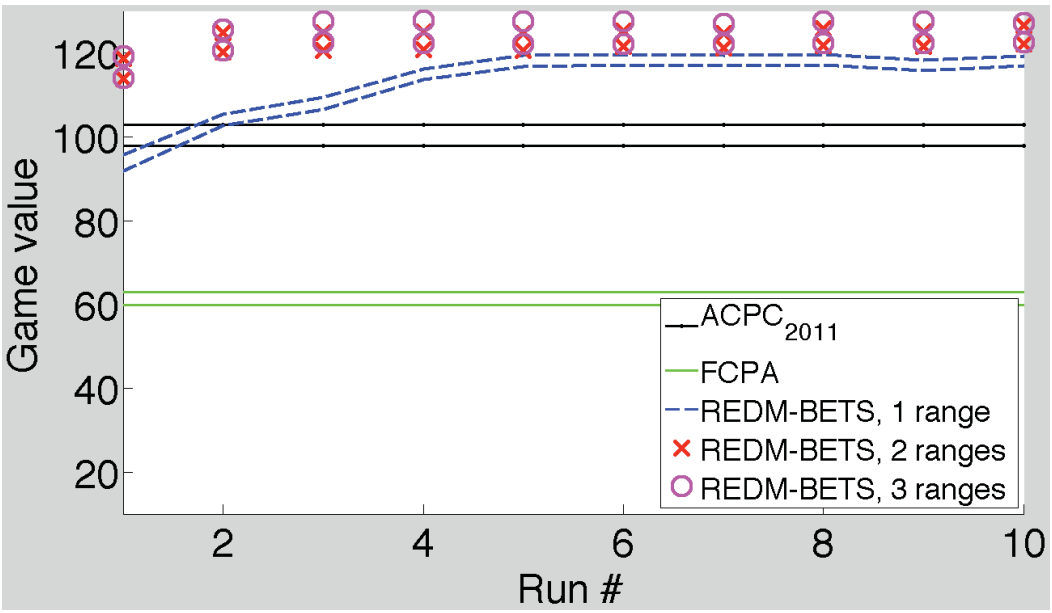


Figure 6.6: Bounds on game values of abstractions using one, two and three default ranges for player two against an $FCPA$ player one, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number.

Looking at these results it is clear that using two ranges instead of one leads to a small improvement in game value. The peak value of the two range runs improved over the peak value of the one range runs by $\approx 23\%$ for player one and $\approx 6\%$ for player two. Additionally, the game value achieved on the first iteration of $REDM$–$BETS$ is much better when using two ranges than it is when using one. This is not surprising, considering that even if both ranges are not of value, it is likely that one of the two bet-sizing players using $[0.5, 0.69]$ and $[1.08, 1.38]$ will be closer to the final bet size than a single bet-sizing player using $[0.71, 0.94]$.

Interestingly, however, the two range and three range curves are virtually identical, both for player one and for player two. As using three ranges creates a larger tree, there is no reason to use a three range abstraction in this case, as we achieve approximately the same game value with a smaller two range abstraction. Furthermore, while these results tell us that using more than one range can lead to a gain in game value, it's not clear that it is valuable to have multiple ranges all throughout the tree. In fact, as I show in the next section, most of the gain in value comes from just a few key extra bets, and identifying where I need these extra bets allows me to create even smaller abstractions while maintaining the game value.

6.6 Pruning the tree

Now I introduce the idea of using the importance metric $R^T_{i,|\cdot|0}$ introduced in Section 5.4 to prune the game tree after a single run of $BETS$, to determine which decision nodes have more than one

bet-sizing player that is valuable enough to keep. At a high level, the idea is to keep only the most important bet-sizing player at each decision node, and then in addition to this add a few extra bet-sizing players that, among the remaining bet-sizing players, have the highest $R^T_{i,|\cdot|0}$ values. I use the parameter e to define the number of extra bet-sizing players to keep. For example, if $e = 1$ I keep only one extra bet-sizing player. If I am pruning based on two ranges, then there will be exactly e decision nodes in the tree with two bet-sizing players to choose from. So for $e = 1$, there will be one decision node at which there are two bet-sizing players.

I implement this in FindImportantBets, Algorithm 8. I create a hash $keepBetPlayers\{\}$ mapping from bet-sizing player histories to a boolean indicating if they will remain in the tree. I initialize this hash as false for all. I now go through the entire tree, and at each decision node with actions leading to bet-sizing players, if bet-sizing player i has the largest $R^T_{i,|\cdot|0}$ value, I set $keepBetPlayers\{h(i)\} = true$. Next, I sort the $R^T_{i,|\cdot|0}$ values of the remaining bet-sizing players, and I set $keepBetPlayers\{\}$ to true for the top e bet-sizing players. Note that the chance of a tie in $R^T_{i,|\cdot|0}$ values is extremely small, as these values are summed over millions of iterations. In the event of a tie I simply keep the first bet-sizing player I encounter.

After running FindImportantBets, I must create a new tree removing all bet-sizing players that are not marked to be kept. This is detailed in PruneTree, Algorithm 9.

Algorithm 8 FindImportantBets

Require: G
Require: e
Require: $Rvalues\{\}$
  // The public action sequence is the sequence of actions known to all players
  // Thus, this does not include the actions of bet-sizing players
  Denote the public action sequence that leads to bet-sizing player i as $a(i)$
  $keepBetPlayers\{\}$, a hash map from public action sequences $a(i)$ to boolean; initially false
  for each node N in G_1 **do**
    if N is a main player node **then**
      $maxR = 0$
      for each bet-sizing player i at N **do**
        if $Rvalues\{a(i)\} > maxR$ **then**
          $maxR = Rvalues\{a(i)\}$
          $keepBetPlayer = a(i)$
        end if
      end for
      if $maxR > 0$ **then**
        $keepBetPlayers\{a(i)\} = true$
        Remove bet-sizing player $keepBetPlayer$ from $Rvalues\{\}$
      end if
    end if
  end for
  Sort $Rvalues\{\}$ from largest to smallest, store sorted $a(i)$ sequences in $sortedExtraBets[]$
  for $extraBet$ from 1 to e **do**
    $keepBetPlayers\{sortedExtraBets[extraBet]\} = true$
  end for
  return $keepBetPlayers\{\}$

Algorithm 9 PruneTree

Require: G
Require: N
  // The public action sequence is the sequence of actions known to all players
  // Thus, this does not include the actions of bet-sizing players
  Denote the public action sequence that leads to bet-sizing player i as $a(i)$
Require: $keepBetPlayers\{\}$, a hash map from public action sequences $a(i)$ to boolean
  for each action a at N **do**
    if a leads to bet-sizing player i **then**
      if $keepBetPlayers\{a(i)\}$ **then**
        Take action a, followed directly by action H
      else
        Remove bet-sizing player i and the sub-tree below from G
      end if
    else
      Take action a
    end if
    Denote the current node N'
    $G = PruneTree(G, N', keepBetPlayers\{\})$
  end for

Once I've finished pruning the tree, I now continue as I did in RED–$BETS$ and $REDM$–$BETS$. I center ranges on the effective bet sizes given by $BETS$, recalculating the range size to maintain the fixed-ratio, and adding and removing bet-sizing players from the end of the tree. Note, however, that I use only the bottom default range to add bet-sizing players from this point on. Recall that, with fixed-ratio ranges, the worst-case all-in bet size depends on the default bet size used to add a new bet-sizing player (see Sections 6.3.1 and 6.3.2). Using the bottom default range to add bet-sizing players is a conservative move, designed to help reduce the size of the worst-case all-in. As I use a single default range to add bet-sizing players from this point forward, this means that there will never be more than e extra bets in the tree.

To support multiple bet-sizing players per decision node, but only add bet-sizing players based on the bottom default range, I must once again modify the AdjustTree algorithm (Algorithm 3, Section 5.7). The new version, AdjustPrunedTree, is shown in Algorithm 10. AdjustPrunedTree is a hybrid of AdjustTree and AdjustTreeMultiBets (Algorithm 7, Section 6.5). Like AdjustTree, it only takes a single default range as input. The loop over bet-sizing players that was added to AdjustTreeMultiBets still exists, but now is used in a more restricted way. AdjustTreeMultiBets loops over all ranges, attempting to add bet-sizing players or calling AdjustRange on existing bet-sizing players. AdjustPrunedTree only attempts to add a bet-sizing player once, as AdjustTree does (using the bottom default range). If bet-sizing players exist already, however, it loops over all of them, calling AdjustRange each time.

I combine all of these steps in the final algorithm of this dissertation: Range Enhanced with Frugal Inclusion of Notable Extra bets and Depth-fixed Bet-size Extraction through Transformation Solving, or $REFINED$–$BETS$. It is detailed in Algorithm 11.

Algorithm 10 AdjustPrunedTree

Require: G
Require: N
Require: L^d
Require: H^d
  if N is an opponent node **then**
    for each bet size b in the opponent abstraction **do**
      if b is not the in tree **then**
        if b is legal **then**
          Add bet b to G at N, and a node below b with fold, call and all-in
        end if
      else
        if b is no longer legal **then**
          remove b and the sub-tree below from G at N
        end if
      end if
    end for
  else if N is not a chance node **then**
    Denote the size of an all-in bet at this decision node as s_{all-in}
    if there is no bet-sizing player at N **then**
      if $H^d < s_{all-in} - \alpha$ **then**
        Add a bet-sizing player to G at N using $[L^d, H^d]$, and an opponent node below this with fold, call and all-in
      end if
    else
      // The addition of this loop is the significant change from $AdjustTree$
      // (Algorithm 3, Section 5.7)
      for each bet-sizing player i at N **do**
        Let L_i, H_i be the L and H values used by bet-sizing player i at N
        $(L_i, H_i, useBet) = AdjustRange(L_i, H_i, \alpha, s_{all-in})$
        if $useBet == false$ **then**
          Remove bet-sizing player i and the sub-tree below from G
        end if
      end for
    end if
  end if
  if N is a chance node **then**
    Sample c from chance
    Take action c
    Denote the current node N'
    $G = AdjustTree(G, N', L^d, H^d, \alpha)$
  else
    for each action a at N **do**
      Take action a
      if a leads to bet-sizing player i **then**
        Take action H_i
      end if
      Denote the current node N'
      $G = AdjustPrunedTree(G, N', L^d, H^d)$
    end for
  end if
  return G

Algorithm 11 The $REFINED–BETS$ algorithm

Require: Multiplayer betting game G'
Require: R runs
Require: T iterations of $BETS$ per run
Require: $numRanges$, the number of default ranges to use
Require: $L^d[]$, an array of size $numRanges$
Require: $H^d[]$, an array of size $numRanges$
Require: α
Require: P^d
Require: e
  Set $G_1 = G'$
  Denote the public action sequence that leads to bet-sizing player i as $a(i)$
  $keepBetPlayers\{\}$, a hash map from public action sequences $a(i)$, to boolean; initially false
  $Rvalues\{\}$, a hash map from public action sequences $a(i)$, to $R^T_{i,|\cdot|0}$; initially 0
  for r from 1 to $R-1$ **do**
    $(A_r, Rvalues\{\}) = BETS(G_r, T)$
    if $r == 1$ **then**
      $keepBetPlayers\{\} = FindImportantBets(G_1, e, Rvalues\{\})$
      Set N to the root node of G_1
      $G_1 = PruneTree(G_1, N, keepBetPlayers\{\})$
    end if
    Create new multiplayer betting game G_{r+1} from G_r as follows:
    for each bet-sizing player i in G_r **do**
      Obtain s_i from A_r
      $w_i = \frac{(1+2s_i)(P^d-1)}{P^d+1}$
      $H_i = s_i + w_i/2$ and $L_i = s_i - w_i/2$
    end for
    Set N to the root node of G_{r+1}
    $AdjustPrunedTree(G_{r+1}, N, L^d[1], H^d[1])$
  end for
  $A_R = BETS(G_R, T)$
  return Set of abstractions A

6.7 Application of $REFINED$–$BETS$ to 5 bucket Texas hold'em

Figures 6.7 and 6.8 show the results of applying $REFINED$–$BETS$ using $e = 1$ and $e = 25$ on players one and two, respectively. Two default ranges are used in the creation of the initial trees that are later pruned. After pruning the tree, I used $T = 45,000,000$ iterations of $BETS$ for $e = 1$ and $e = 25$ for both players, as I did for the full two ranges. For comparison, the game value curves for $REDM$–$BETS$ using one and two default ranges from Figures 6.5 and 6.6 are reproduced on these graphs. The game value curves for three default ranges are omitted, as they are almost identical to the two range curves (see Section 6.5.2).

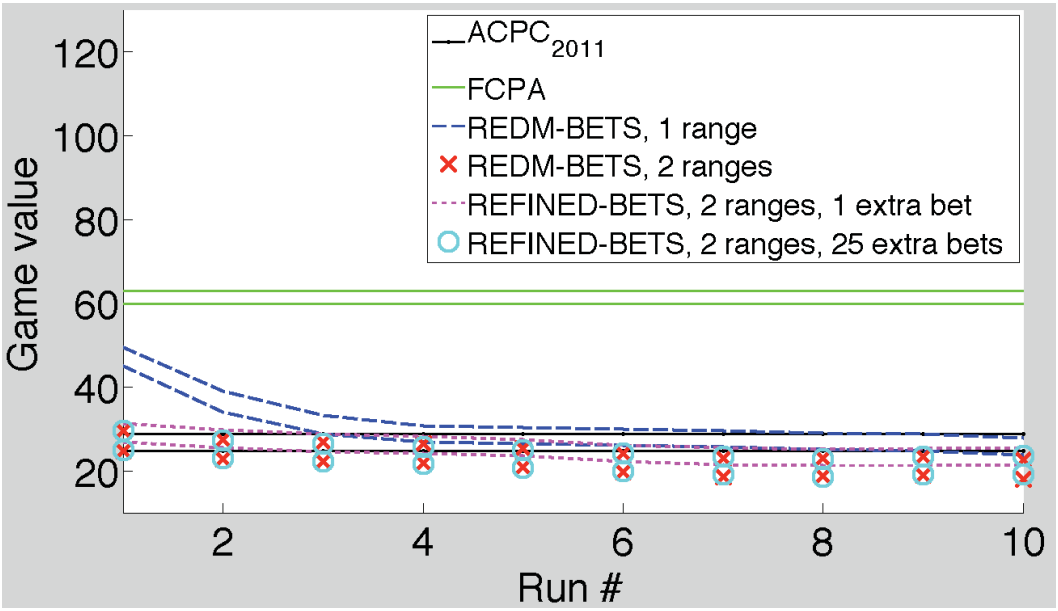


Figure 6.7: Bounds on game values of abstractions using one and two default ranges for player one against an $FCPA$ player two, in a 5 bucket perfect recall card abstraction of Texas hold'em. The pruned runs start with two default ranges, keeping $e = 1$ and $e = 25$ extra bets. The Y-axis is value to player two, so player one wishes to decrease this number.

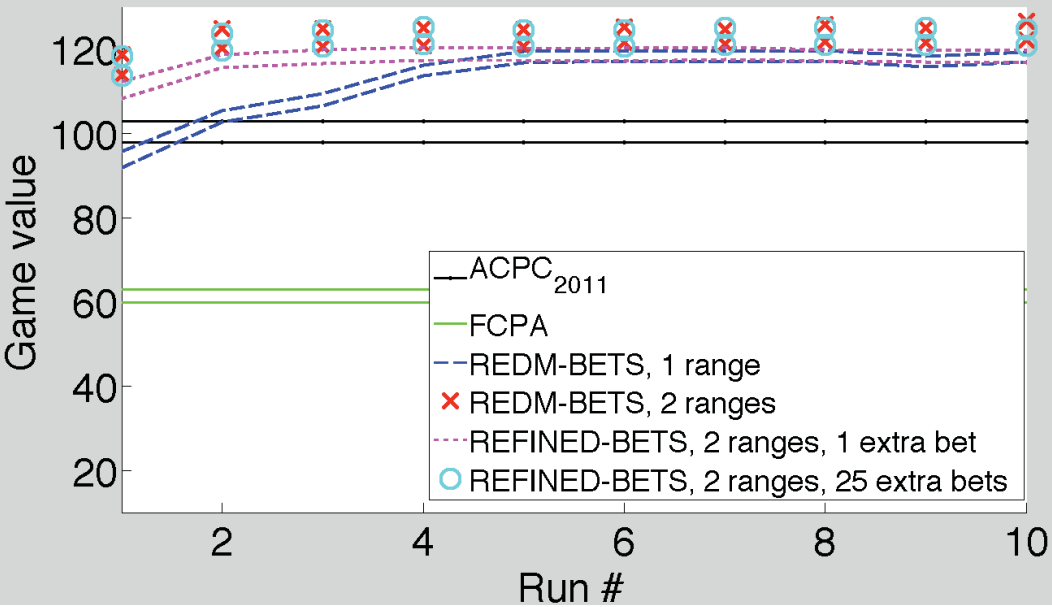


Figure 6.8: Bounds on game values of abstractions using one and two default ranges for player two against an $FCPA$ player one, in a 5 bucket perfect recall card abstraction of Texas hold'em. The pruned runs start with two default ranges, keeping $e = 1$ and $e = 25$ extra bets. The Y-axis is value to player two, so player two wishes to increase this number.

6.7.1 Analysis of performance

Looking at Figures 6.7 and 6.8, we can immediately see an interesting phenomenon on run $r = 1$. At this point, for player one (Figure 6.7) the value of the two range abstraction and both pruned abstractions are very similar, and both have significantly greater value than the one range abstraction. For player two (Figure 6.8) the $e = 25$ abstraction again equals the value of the two range abstraction, but $e = 1$ does not perform quite as well as it does for player one. The $e = 1$ abstraction still has a better game value than the one range abstraction, but lags behind the value of the rest of the abstractions. It is not particularly surprising that the two range abstractions immediately perform better than the one range abstractions, as there are two bets to choose from at most decision nodes.[1] The fact that the $e = 1$ abstraction has significantly better game value on $r = 1$ than the one range abstraction in both cases, however, shows a big advantage of the pruning process. There are more ranges to choose from initially, and so the remaining bet-sizing players end up much closer to their target values after just one run when compared to the bet-sizing players in the one range abstractions.

Once the curves level off for player one (Figure 6.7), the value of the $e = 1$ abstractions end up approximately half way between the value of the two range abstractions and the one range abstractions. This is a notable result; half of the value gained by using two ranges *everywhere* can be gained by using two ranges exactly *once*. Furthermore, using $e = 25$ makes up the remainder of this gap, obtaining approximately the same game value as having two ranges everywhere. For player two, the

[1] At some decision nodes where the pot size is very large an $H_1^d = 0.69$ bet will be legal and an $H_2^d = 1.38$ will not

$e = 1$ abstractions level off to the same value as the one range abstractions, and again the $e = 25$ abstractions equal the value of the two range abstractions.

To understand why the $e = 1$ abstractions outperform the one range abstractions for player one, while providing no advantage at the peak point over one range for player two, I consider the bet sizes of the extra bet in both cases. For player one, the extra bet is added directly after the opponent chooses to call the big blind. At $r = 5$, the bet-sizing player using the lower range chooses a value of 0.94 while the upper range player chooses 1.71. The strategy created with CFR using this action abstraction uses both of these bet sizes, betting 0.94 26.5% of the time, and 1.71 16.5% of the time. The same action sequence in the one range case is using a bet size of 1.28 at $r = 5$. The strategy created with CFR using this action abstraction makes this bet 40% of the time. For player two, the extra bet is added as the very first action of the game. At $r = 5$, the lower range player chooses a value of 0.59 while the upper range player uses 0.66. These bets are made 53% and 30% of the time, respectively. In the one range case a bet of size 0.63 is chosen here at $r = 5$. This bet is made 84% of the time. The bet sizes chosen by all the bet-sizing players I have mentioned here remain relatively constant for the next 10 runs of $BETS$ in each case.

The contrast between these two examples is notable. When a second bet-sizing player is added for player one, the result is that this new player and the other bet-sizing player at the same decision node choose very different bet sizes, both a significant distance from the bet size chosen if no extra player is added, which falls in between them. In the case of player two, however, the extra bet-sizing player chooses a bet size very similar to that chosen by the other bet-sizing player at the same decision node. The bet size chosen at this point when no extra player is added is, again, in between these two values. The result for player one is a notable increase in game value, while for player two if there is an increase in game value it is insignificant.

This is not an isolated incident. If the $e = 25$ runs are extended to $r = 15$, and bets within 0.1 of each other are considered to be merged, there are only $e = 16$ remaining extra bet-sizing players for player one and $e = 18$ remaining extra bet-sizing players for player two. There are a couple of things to note about this result. First, the fact that many bet-sizing players converge on very similar values when initialized with different ranges suggest that local maxima are not causing a significant problem for those players. Second, this illustrates a limitation of pruning the bet-sizing players on run $r = 1$. Until completing many runs of $BETS$, there is no way to know which bet-sizing players will end up very close to one another and which ones will be separated by a significant amount. On a practical level, the fact that abstraction sizes tend to increase over time, first mentioned in Section 6.1, dictates that I should only run until I hit the peak game value and no longer, as the abstractions tend to get bigger after this point without adding value. The increases in abstraction size dwarf the memory savings of removing duplicate bet sizes. It typically takes $10 - 15$ runs for bets to merge, and the game value curves usually level off well before that. For this reason I simply overestimate the number of extra bets e that will actually be needed, and I do not try to do any duplicate detection

or removal.

It should be noted that another possible option would be to complete many runs of $BETS$ to determine which bet sizes eventually merge. I could then take the abstraction created at the point where the game value curve levels off, and for each group of bet-sizing players that merged together on a later run, choose just one bet-sizing player to keep and remove the rest. This would reduce the size of the tree while, presumably, maintaining the peak game value.[2] This would take significant computational time for small memory savings, and thus would only be worthwhile if memory resources are extremely limited.

6.7.2 Size comparison

Now I compare the sizes of the one, two and three range abstractions along with the pruned two range abstractions. For the two range abstraction on run $r = 5$ there are $e = 585$ and $e = 706$ extra bets for players one and two, respectively. Recall that the pruned trees have e extra bets after the pruning step on run 1, and continue to have e extra bets from then on, as from this point forward we only add bets using the bottom default range (see Section 6.6). To put these difference in terms of tree size, I show the (information set, action) pairs of the one range, $e = 1$, $e = 25$, two range and three range abstractions for both players one and two on run $r = 5$ in Tables 6.2 and 6.3, respectively.

Abstraction	# (I, a) pairs
One range ($e = 0$)	3,217,970
Start with two ranges, $e = 1$	4,678,670
Start with two ranges, $e = 25$	7,104,940
Two ranges, no pruning ($e = 585$)	14,585,790
Three ranges, no pruning ($e = 1470$)	28,734,090

Table 6.2: Number of (I, a) pairs in various action abstractions for player one, generated using $REDM$–$BETS$ and $REFINED$–$BETS$. In each case $BETS$ was run five times. In all cases the opponent uses $FCPA$, and a 5 bucket perfect recall card abstraction of Texas hold'em was used.

Abstraction	# (I, a) pairs
One range ($e = 0$)	3,614,400
Start with two ranges, $e = 1$	5,223,850
Start with two ranges, $e = 25$	7,591,300
Two ranges, no pruning ($e = 706$)	17,906,500
Three ranges, no pruning ($e = 1585$)	30,797,090

Table 6.3: Number of (I, a) pairs in various action abstractions for player two, generated using $REDM$–$BETS$ and $REFINED$–$BETS$. In each case $BETS$ was run five times. In all cases the opponent uses $FCPA$, and a 5 bucket perfect recall card abstraction of Texas hold'em was used.

Examining these tables I note two things. First of all, the $e = 1$ abstractions are $\approx 45\%$ larger than the one range abstractions in both cases. This is due to the fact that the extra bet-sizing player

[2] This is not guaranteed — it's possible that additional runs may then be required to reach the peak point once more

is added very early in the tree in both cases (see Section 6.7.1), creating a large sub-tree beneath it. So choosing $e = 1$ increases the size of the abstractions greatly, while only providing an increase in game value in the case of player two. The $e = 25$ abstractions are another $\approx 52\%$ and $\approx 45\%$ larger than the $e = 1$ abstractions for players one and two, respectively. This is still less than half the size of the two range abstractions for both players, and about one quarter the size of three range abstractions. As we saw in Section 6.7.1, however, the $e = 25$ abstractions obtain approximately the same game value as these larger abstractions that do not use pruning.

This clearly quantifies the advantage I gain by pruning — by using $e = 25$, I reduce the tree by a factor of 2 from two default ranges, or 4 from three default ranges, while maintaining the game value. If pruning is to be used, however, it is important to choose a value of e that is large enough to capture the most important extra bets, and not just extra bets that will merge with ones that already exist while making the tree larger. While $e = 25$ is enough to equal the value of two (and three) ranges in this game, the number of extra bets needed varies as the card abstraction and opponent abstraction change. I discuss this further in the next chapter. Additionally, for 5 bucket perfect recall Texas hold'em with an $FCPA$ opponent I pruned based on two default ranges, as two and three default ranges obtain approximately equivalent game value. The pruning process can be applied with any number of default ranges; in the next chapter I will show examples where I prune based on three default ranges.

6.8 Kuhn revisited

In this section I show that $REFINED$–$BETS$ can help to avoid local maxima in half-street Kuhn poker (see Chapter 4), without prior knowledge of the global maxima. As I showed in Chapter 4, running $BETS$ with one range converges to the optimal value of 0.414 in half-street Kuhn poker as long as $0 \leq L \leq 0.414$ and $0.414 \leq H < 1$. If $L > 0.414$ and $H < 1$, then $BETS$ will converge to L. If $H < 0.414$, $BETS$ will converge to H (See Section 4.4).

A problem arises when $H > 1$. The $BETS$ algorithm may converge to a bet size $s \geq 1$, because all bet sizes greater than one are part of a local maxima. If $s \geq 1$ then the Nash equilibrium strategy is for the betting player to stop bluffing, and the responding player stops calling with their Queen. As a result, both L and H have the same value, and the bet sizing player cannot gain value by changing their strategy. I introduced this problem in Section 4.5 and showed the game value of half-street Kuhn for bet sizes from 0 to 2. I reproduce that figure again here as Figure 6.9 for convenience.

When $REFINED$–$BETS$ is applied to this game, as long as one of the initial ranges has $H < 1$, we can avoid this local maxima problem and, given enough runs of $BETS$, arrive at the optimal bet size of 0.414. I applied $REFINED$–$BETS$ with the three default ranges I use throughout this chapter, $[0.5, 0.69]$, $[1.08, 1.38]$ and $[1.95, 2.42]$. Running for $500,000$ iterations, as I did in Section 4.4, I kept only the bet with the highest importance value after the first run. This led to keeping the bottom range, with a bet size of 0.5. This range has the highest importance value

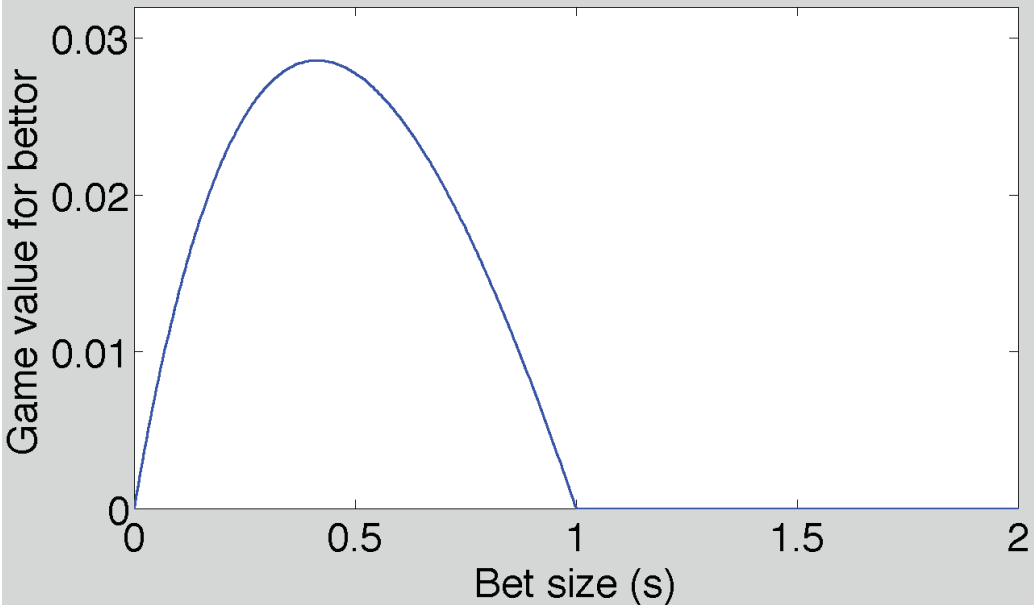


Figure 6.9: Half-street Kuhn poker game value per bet size, extended to 2 pot.

because both 0.5 and 0.69 have higher utility than bets in the other two ranges (see Figure 6.9). On the next run this bet sizing player converged to within 1% of 0.414, the optimal value.

Repeating this experiment using ranges $[0.1, 0.2]$, $[0.5, 0.69]$ and $[0.71, 0.94]$, and again pruning after run 1, the second range was the only remaining bet sizing player. In this case the bottom range chooses 0.2, the top range chooses 0.71 and the middle range chooses 0.5. As we can see from Figure 6.9, 0.5 has the highest game value of these, and thus the highest importance value. Once more the algorithm converged to 0.414 on the next run. This showcases the power of $REFINED\text{–}BETS$ - if we have at least one range in the region of the global maxima, we can avoid local maxima. Additionally, we arrive at the correct answer very quickly if we use many ranges on the first run, as pruning using the importance values correctly picks the bet-sizing player closest to the global maxima.

Chapter 7

Changing the card and opponent abstractions

In this chapter I apply the $REDM$–$BETS$ and $REFINED$–$BETS$ algorithms to imperfect recall card abstractions and two different opponent action abstractions. I begin by describing how I evaluate action abstractions paired with imperfect recall card abstractions (Section 7.1). Next, I apply $REDM$–$BETS$ and $REFINED$–$BETS$ from Chapter 6 to a small imperfect recall card abstraction of no-limit Texas hold'em in Sections 7.2.1 and 7.2.2. For these experiments I continue to use an $FCPA$ opponent action abstraction. I discuss how using a larger card abstraction leads to very little change in the action abstractions created in Section 7.3. This leads me to the idea of creating action abstractions using small card abstractions, then pairing these action abstractions with larger card abstractions, which I explore in Section 7.4. In the next section (7.5) I consider the effect of changing the opponent action abstraction to the $ACPC_{2011}$ abstraction (first introduced in Section 5.8.1). Finally, in Section 7.6, I describe how an agent was created using one of my action abstractions paired with a very large card abstraction, and show how this agent compares to other top no-limit Texas hold'em agents.

7.1 Evaluation in imperfect recall games

In Chapters 5 and 6 I used a 5 bucket perfect recall card abstraction of Texas hold'em in my experiments. The advantage of using a perfect recall card abstraction is that I can calculate best response values within that card abstraction (combined with the action abstraction I'm using), and thus obtain bounds on the possible values of a given (action abstraction, card abstraction) pair. When using imperfect recall card abstractions things are more complicated, as computing a best response is intractable [67]. While imperfect recall makes evaluation more difficult, it is generally considered to create better performing card abstractions in comparison with perfect recall in no-limit Texas hold'em ([32, 42, 67]). For this reason I wish to use imperfect recall card abstractions, and so in this section I describe my methodology for evaluating them.

I start the evaluation process, as before, by creating an ϵ-Nash equilibrium strategy profile for the abstraction I wish to evaluate. The next step when working with perfect recall card abstractions is to determine absolute bounds on the value of the action abstraction within the given card abstraction. Instead, a large number of hands of self-play are dealt, in the full no-limit game, with this strategy profile. For a given number of hands, a 95% confidence interval on the result of the self-play match can be calculated, assuming a normal distribution for the results of each match. Enough hands are played such that this confidence interval is approximately the same width as the absolute bounds were when using best responses on the perfect recall card abstractions. I found that achieving this level of confidence required playing $1,200,000,000$ hands, leading to 95% confidence intervals of approximately ± 1.1 milli-big-blinds. Across a range of different agents the confidence intervals remained approximately the same width. I use these empirical bounds when creating the game value curves for the action abstractions generated using imperfect recall card abstractions.

To see how these empirical bounds compare to the absolute bounds I calculate for perfect recall games, I performed this procedure on the abstractions created in Section 6.4. These abstractions were generated using a 5 bucket perfect recall game, with an $FCPA$ opponent action abstraction. They were generated by applying $REDM–BETS$ to one default range of $[0.71, 0.94]$. I ran self-play matches of $1,200,000,000$ hands for each of the first 10 runs of $REDM–BETS$ for both player one and player two. The resulting 95% confidence intervals on the bounds of the game value are shown in Figures 7.1 and 7.2, along with the absolute bounds.

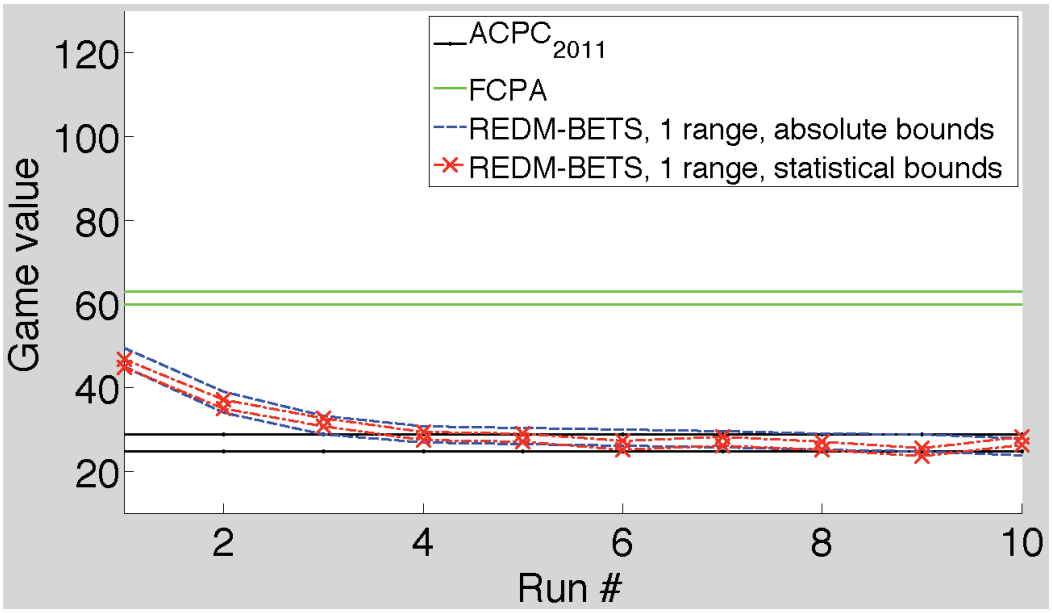


Figure 7.1: Absolute bounds compared with empirical 95% confidence intervals on the game value of abstractions created using $REDM–BETS$ for player one against an $FCPA$ player two, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to decrease this number.

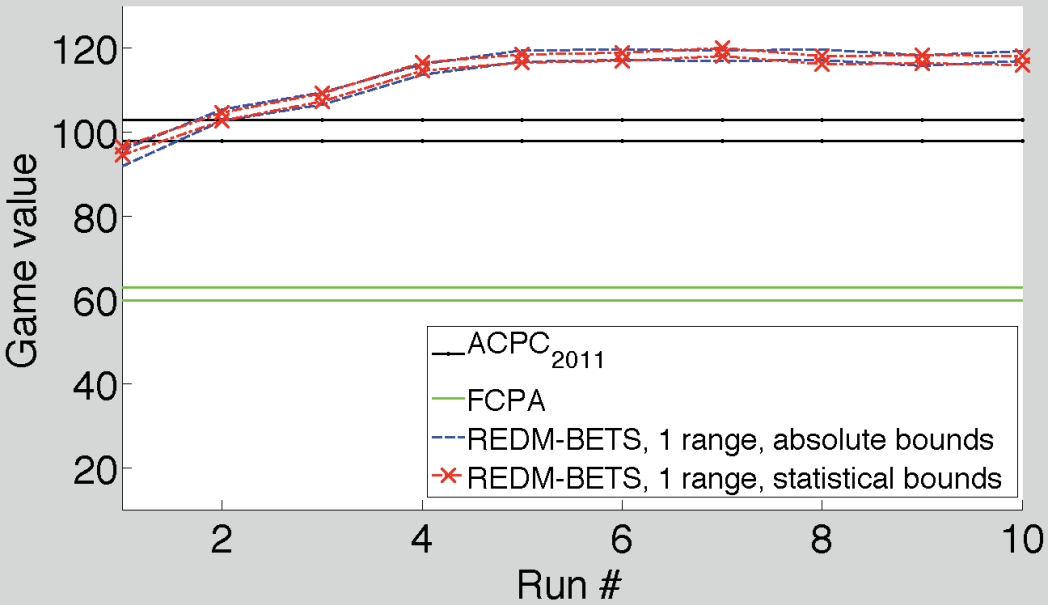


Figure 7.2: Absolute bounds compared with empirical 95% confidence intervals on the game value of abstractions created using $REDM\text{–}BETS$ for player two against an $FCPA$ player one, in a 5 bucket perfect recall card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number.

Looking at both figures it is clear that playing $1,200,000,000$ hands leaves a small amount of variance in the game value, small enough that the general shape of the curve remains the same. This allows my analysis to proceed as it did when using the absolute bounds. Throughout the remainder of this chapter I work with imperfect recall card abstractions, and thus I cannot generate the absolute bounds. Instead I will use the empirical bounds in their place, generated each time by running $1,200,000,000$ hands of self-play in the full no-limit game.

7.2 Results in $IR - 100$

I now repeat the experiments of Sections 6.5.2 and 6.7 with an imperfect recall card abstraction. All imperfect recall card abstractions I use in this chapter were generated using the k-means clustering method of Johanson et al. [42] mentioned in Section 3.7.1. I always use 169 buckets pre-flop, as that is the number of unique pre-flop hands in Texas hold'em. I use the convention $IR - Ns$ to refer to an imperfect recall card abstraction with N buckets on the flop, turn and river. If the number of buckets on each of these rounds is different, I specify $IR - Ns$ on the flop, $IR - Ms$ on the turn, and so forth, where N and M are numbers of buckets. In this Section I use an $IR - 100$ card abstraction, and I continue to use an $FCPA$ opponent action abstraction.

7.2.1 *REDM–BETS*

For my first set of results I apply $REDM–BETS$ to this game using one, two and three default ranges. Following the methodology outlined in Section 6.1, for one default range I used $T = 60,000,000$, for two default ranges I used $T = 70,000,000$ and for three default ranges I used $T = 80,000,000$. I use the same default ranges from the previous chapter, as detailed in Table 6.5. Figures 7.3 and 7.4 show the results of 10 runs of $BETS$ for players one and two, respectively.

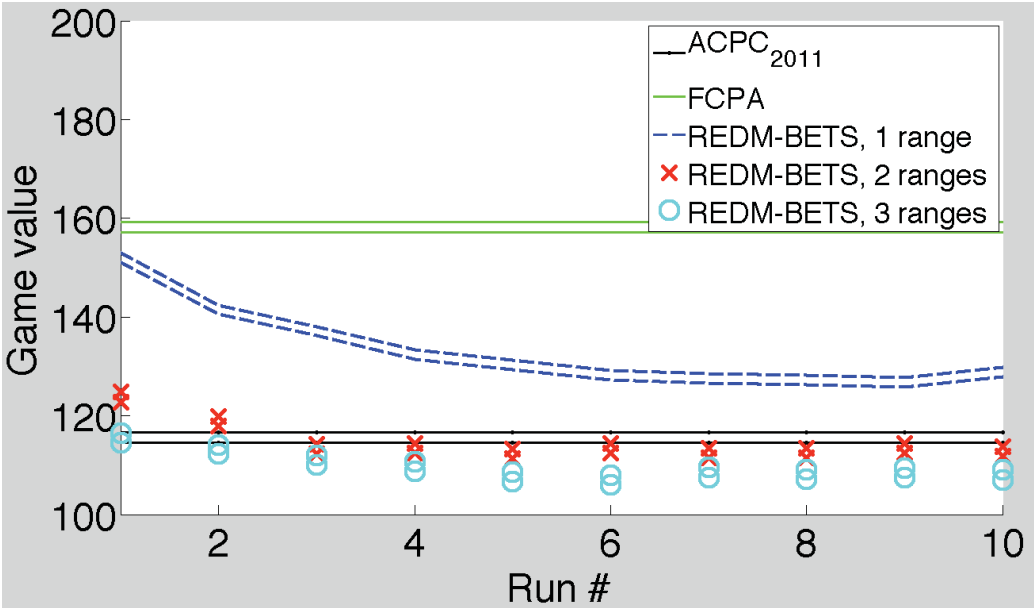


Figure 7.3: Bounds on game values of abstractions using one, two and three default ranges for player one against an $FCPA$ player two, in an $IR - 100$ card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to decrease this number.

The most notable feature of these two graphs is that the game values are significantly larger in magnitude than they were in the 5 bucket perfect recall game. Figures 6.5 and 6.6 show that in the case of a 5 bucket perfect recall game, the game value (for player two) ranged from about 20 milli-big blinds to 125 milli-big blinds, depending on if this value was being minimized (player one) or maximized (player two). In $IR - 100$ the values range from 105 to 200. The difference in scale is due to the fact that in the perfect recall game, there is no way for players to distinguish the top 20% of hands that are in the top bucket. This leads to a significant number of all-ins in the pre-flop, with hands as weak as a King and an 8 of different suits. The effect of this is to reduce the number of decisions made throughout the game, and thus player two has less opportunities to capitalize on the advantage they gain by acting after player one. This problem of not being able to distinguish relatively good hands from the best hands is one of the reasons imperfect recall abstractions are generally preferred over perfect recall abstractions in no-limit Texas hold'em [32, 67].

Looking at the shapes of Figures 7.3 and 7.4 two other differences from the perfect recall 5

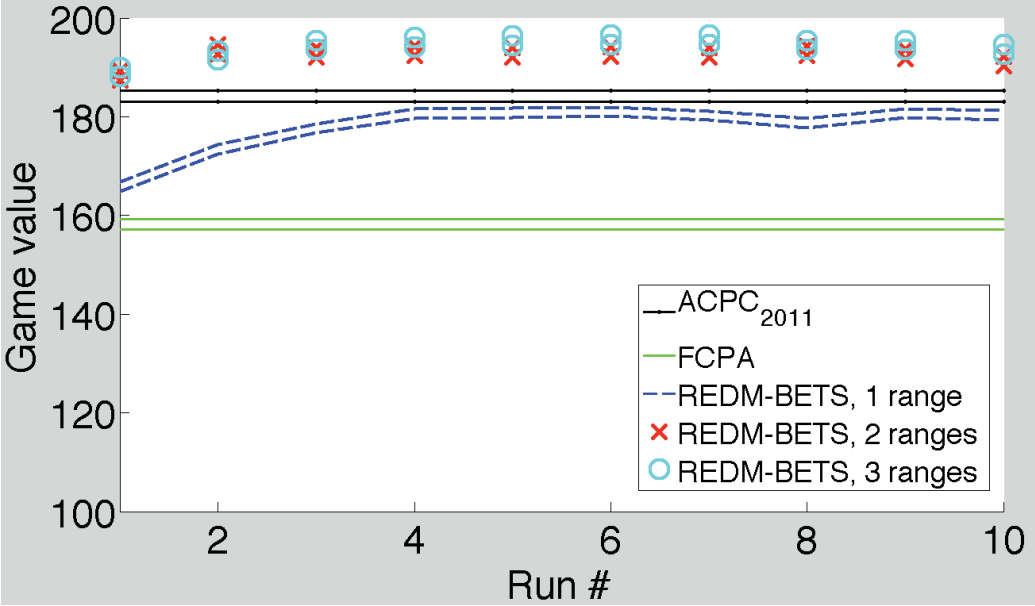


Figure 7.4: Bounds on game values of abstractions using one, two and three default ranges for player two against an $FCPA$ player one, in an $IR - 100$ card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number.

bucket game stand out. First, one range does not do quite as well when compared with $ACPC_{2011}$. In Figures 6.5 and 6.6 we saw that one range achieves a game value that roughly ties $ACPC_{2011}$ for player one and surpasses it for player two by $\approx 18\%$. In $IR - 100$ one range continues to easily outperform $FCPA$, however it does not do as well as $ACPC_{2011}$ for either player. The second difference between the two games is that in $IR - 100$ adding a third range leads to a small increase in game value over two ranges, while in the 5 bucket perfect recall game adding a third range made no difference.

Overall it seems that the number or ranges matters more to game value in this imperfect recall game than it did in the 5 bucket perfect recall game. Next I consider how this impacts pruning of the tree.

7.2.2 $REFINED$–$BETS$

As three ranges performs better than two, I applied $REFINED$–$BETS$ to this $IR - 100$ game by pruning the three range abstraction. This means that, given $e > 1$ extra bet-sizing players, there may be places in the tree where two of those extra bet-sizing players are added at the same decision node, making a total of three bet-sizing players at that node. Once again I show the results of using $e = 1$ and $e = 25$, as I did in Section 6.7 for the 5 bucket perfect recall game (Figures 6.7 and 6.8). I used $T = 65,000,000$ for $e = 1$ and $T = 80,000,000$ for $e = 25$. Recall from Section 7.2.1 that $T = 80,000,000$ was used for the full three ranges as well.

The results of applying $REFINED$–$BETS$ with three default ranges to $IR - 100$ are given

in Figures 7.5 and 7.6.

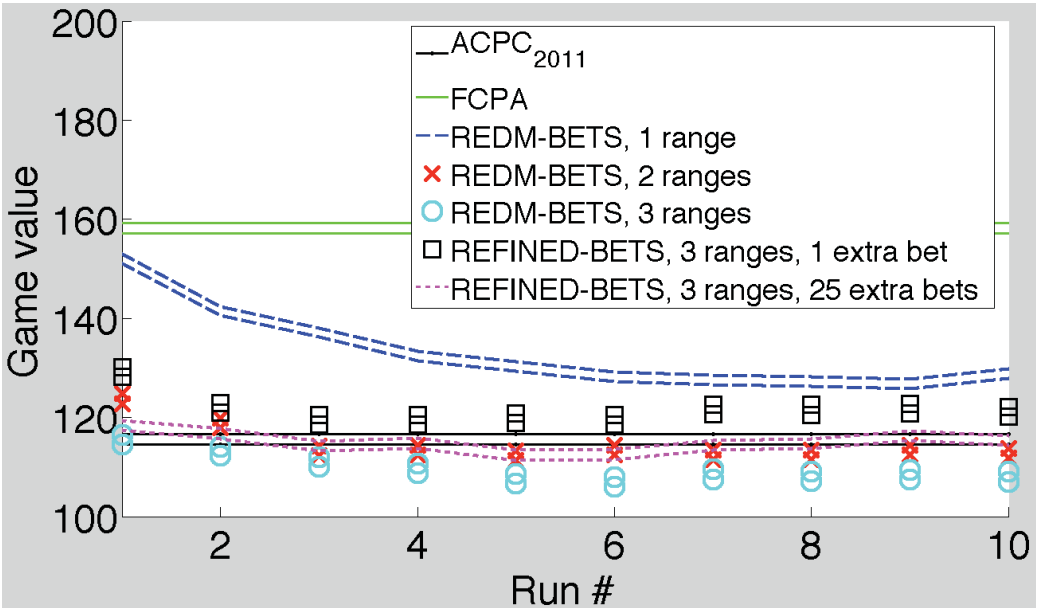


Figure 7.5: Bounds on game values of abstractions using one, two and three default ranges for player one against an $FCPA$ player two, in an $IR - 100$ card abstraction of Texas hold'em. The pruned runs start with three default ranges, keeping $e = 1$ and $e = 25$ extra bets. The Y-axis is value to player two, so player one wishes to decrease this number.

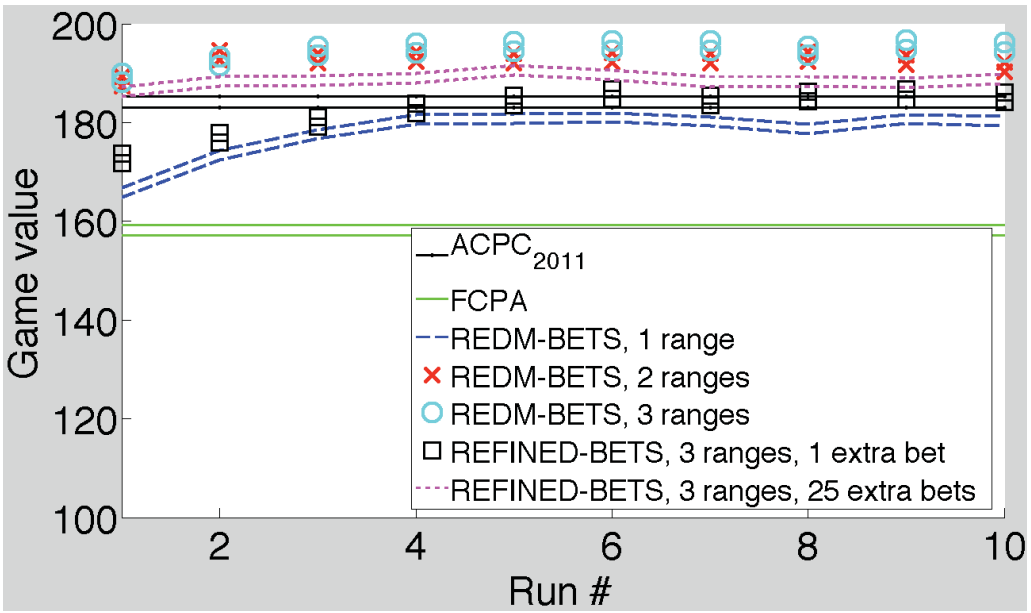


Figure 7.6: Bounds on game values of abstractions using one, two and three default ranges for player two against an $FCPA$ player one, in an $IR - 100$ card abstraction of Texas hold'em. The pruned runs start with three default ranges, keeping $e = 1$ and $e = 25$ extra bets. The Y-axis is value to player two, so player two wishes to increase this number.

Looking at Figure 7.5, we see that $e = 1$ improves upon the performance of one default range for player one, as was the case in the 5 bucket perfect recall game. While in the 5 bucket perfect recall game the extra bet was added in the pre-flop after player two checks, here it's added in the pre-flop after player two bets. At that point bets of size 0.93 and 2.47 are used. In comparison, player one uses a bet size of 1.13 at this point in the one range case. The gains of $e = 1$ over one range are comparable to the gains of $e = 25$ over $e = 1$. The bounds of $e = 25$ and two ranges are very similar on runs 5 and 6, and while two ranges stabilizes, $e = 25$ trends upwards a bit. Due to the empirical nature of the bounds it's hard to draw conclusions about small changes in the value. It is of note, however, that many of these curves tend to uptick slightly in the last few runs, which is possibly due to the phenomenon described in Section 6.1 — after each short run the games tend to get larger, and thus using the same number T of iterations of $BETS$ leads to slightly less accurate bet sizes. Even if we ignore this upward trend in the last few runs we can see that $e = 25$ does not perform as well as three full ranges for player one.

For player two $e = 1$ improves over the performance of one range, as shown in Figure 7.6. This is an improvement over the 5 bucket perfect recall game when $e = 1$ did not add any value. Notably, $e = 1$ is enough to equal the value of $ACPC_{2011}$, averaging a slightly greater value after the curve levels off, with empirical bounds overlapping. The bet is added at the same point in the game as in the perfect recall case — as the first action made. In the imperfect recall game the $e = 1$ case keeps the first and second default ranges, setting them to values of 0.78 and 0.93. This is different from the perfect recall game where bet sizes of 0.59 and 0.66 are used. For the $e = 25$ pruning run the value does not improve a great deal over that of $e = 1$, falling below the value of two ranges.

In light of the fact that 25 extra bets is outperformed in both cases by the full three ranges, I repeated the experiment with $e = 50$ and $e = 100$, the results of which are shown in Figures 7.7 and 7.8. For player one we can see a minor improvement over $e = 25$, with both $e = 50$ and $e = 100$ having very similar values. Their value curves beat those of two ranges on average by a very small margin, within the empirical bounds. In player two's case, again there is an improvement over $e = 25$, and while it is a larger improvement than for player one, the value still falls short of three full ranges. $e = 100$ beats the value of $e = 50$ by a very small margin on average, again within the empirical bounds. Overall, the pruned runs are only improving in game value by a marginal amount as we double the number of extra bets. This is very different from the perfect recall game where two ranges pruned with just $e = 25$ obtained the same value as three ranges everywhere.

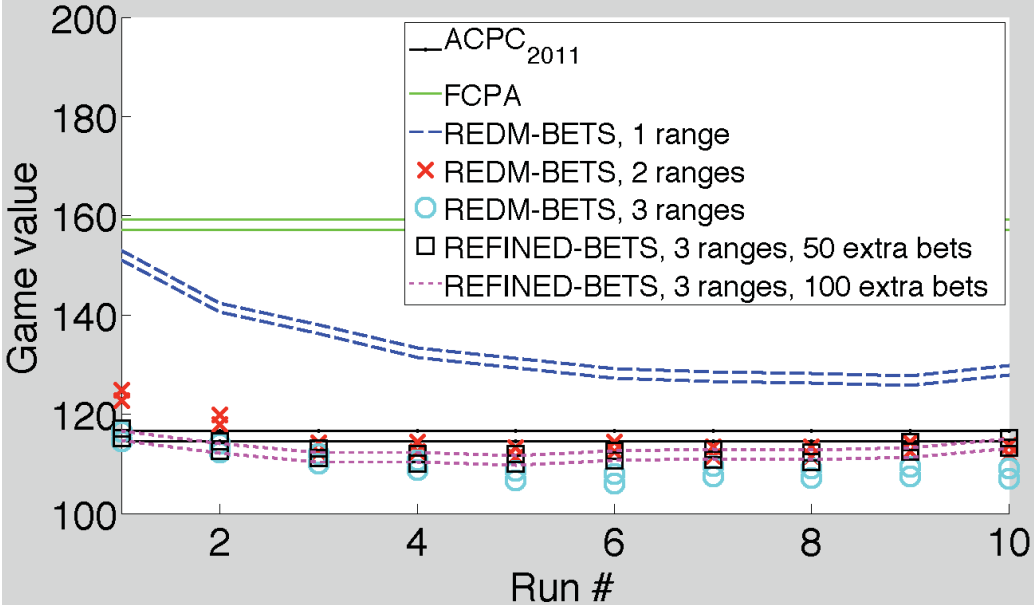


Figure 7.7: Bounds on game values of abstractions using one, two and three default ranges for player one against an $FCPA$ player two, in an $IR - 100$ card abstraction of Texas hold'em. The pruned runs start with three default ranges, keeping $e = 50$ and $e = 100$ extra bets. The Y-axis is value to player two, so player one wishes to decrease this number.

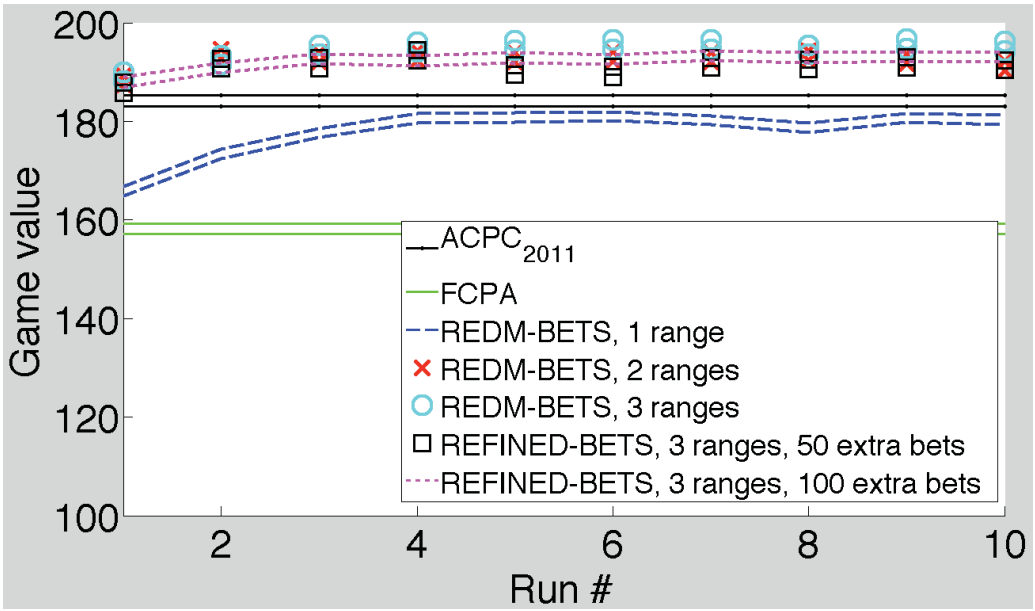


Figure 7.8: Bounds on game values of abstractions using one, two and three default ranges for player two against an $FCPA$ player one, in an $IR - 100$ card abstraction of Texas hold'em. The pruned runs start with three default ranges, keeping $e = 50$ and $e = 100$ extra bets. The Y-axis is value to player two, so player two wishes to increase this number.

7.2.3 Size comparison

Here I repeat the size comparison I did in Section 6.7.2, but applied to this $IR - 100$ game. As before, I use run $r = 5$ to compare the abstractions. The number of extra bets, along with the number of (information set, action) pairs, are shown in Tables 7.1 and 7.2.

Abstraction	Value to player two	# (I, a) pairs
One range ($e = 0$)	130.2 ± 1.01	610,016
Start with three ranges, $e = 1$	119.9 ± 1.05	779,458
Start with three ranges, $e = 25$	112.2 ± 0.99	1,681,840
Start with three ranges, $e = 50$	111.3 ± 1.03	1,938,254
Start with three ranges, $e = 100$	110.7 ± 1.03	2,388,896
Two ranges, no pruning ($e = 547$)	112.3 ± 1.11	2,701,068
Three ranges, no pruning ($e = 1203$)	107.7 ± 0.97	4,726,748

Table 7.1: Number of (I, a) pairs and game value in various action abstractions for player one, generated using $REDM$–$BETS$ and $REFINED$–$BETS$. In each case $BETS$ was run five times. In all cases the opponent uses $FCPA$, and an $IR - 100$ card abstraction of Texas hold'em was used.

Abstraction	Value to player two	# (I, a) pairs
One range ($e = 0$)	180.8 ± 0.98	636,016
Start with three ranges, $e = 1$	184.5 ± 1.00	1,001,686
Start with three ranges, $e = 25$	190.7 ± 0.98	1,988,482
Start with three ranges, $e = 50$	190.4 ± 1.06	2,264,124
Start with three ranges, $e = 100$	192.9 ± 0.96	2,616,552
Two ranges, no pruning ($e = 576$)	193.1 ± 0.97	3,199,296
Three ranges, no pruning ($e = 1516$)	195.5 ± 1.02	6,086,776

Table 7.2: Number of (I, a) pairs and game value in various action abstractions for player two, generated using $REDM$–$BETS$ and $REFINED$–$BETS$. In each case $BETS$ was run five times. In all cases the opponent uses $FCPA$, and an $IR - 100$ card abstraction of Texas hold'em was used.

For player one, $e = 1$ is only $\approx 28\%$ bigger than one range, as compared to $\approx 57\%$ bigger for player two, while it was $\approx 45\%$ bigger for both players in the 5 bucket perfect recall game. In fact, each of the abstractions for player two are larger than the player one equivalents, something that was true for the 5 bucket perfect recall game in all cases except $e = 1$ (see Tables 6.2 and 6.3). This trend continues when I switch to an $ACPC_{2011}$ opponent action abstraction (see Section 7.5).

The size-to-value ratio of these abstractions is not as simple as it was for the 5 bucket perfect recall game. Pruning is still advantageous — the value of $e = 50$ ties or beats two range for both players, and is 72% of the size of two ranges for player one and 71% of the size for player two. Which of these abstractions is considered best, however, depends on how much memory one is willing to trade for diminishing returns in game value. If beating the $ACPC_{2011}$ abstraction's value is considered to be a requirement, then I would want to use at least $e = 50$ for player one and at least $e = 25$ for player two. If I wish to beat the value of two ranges by a significant amount, I can either use three full ranges, or use a larger value of e to create a game between the sizes of the $e = 100$

abstractions the full three range abstractions ($e = 1516$) that has between their values as well. I will consider this trade-off further in Section 7.5.

7.3 Scaling the card abstraction

As $IR - 100$ proved to be a very different card abstraction from the 5 bucket perfect recall card abstraction used in Chapters 5 and 6, it is natural to wonder what happens when the number of imperfect recall buckets is increased. Thus, I applied $REDM$–$BETS$ to an $IR - 570$ abstraction, generated using the same technique that was used to create the $IR-100$ abstraction, and maintaining the $FCPA$ opponent action abstraction. The reason 570 buckets are used is actually a relic from limit Texas hold'em, where an $IR - 570$ abstraction is approximately the same size as a 5 bucket perfect recall abstraction. In no-limit Texas hold'em $IR - 570$ with an $FCPA$ action abstraction is $\approx 25\%$ larger than the 5 bucket perfect recall game with the same $FCPA$ action abstraction.

I used $T = 100,000,000$ for one default range, $T = 130,000,000$ for two default ranges, and $T = 160,000,000$ for three default ranges. Figures 7.9 and 7.10 show the results of $REDM$–$BETS$ applied to this card abstraction using 10 runs of $BETS$.

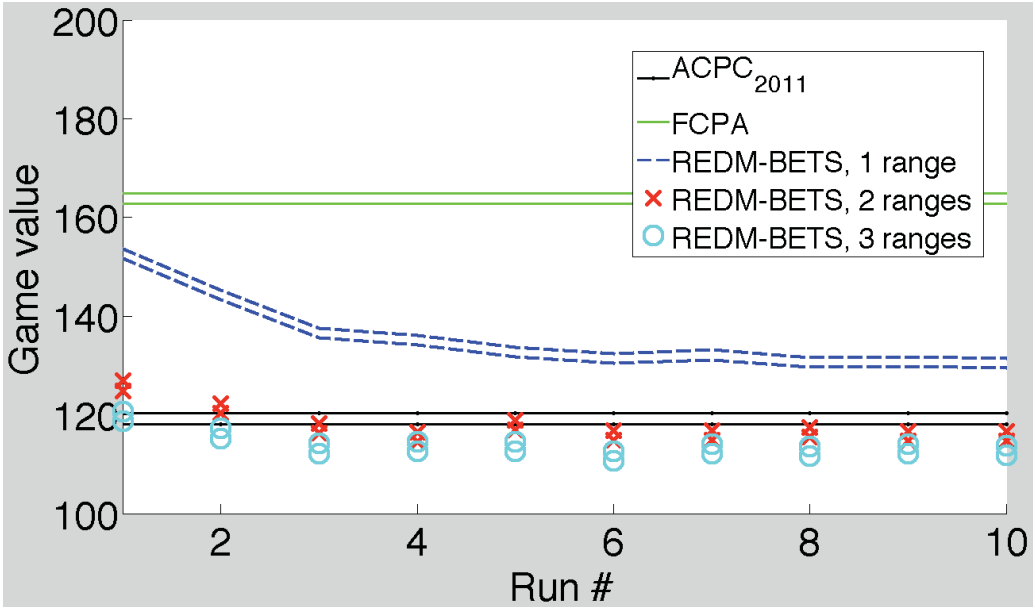


Figure 7.9: Bounds on game values of abstractions using one, two and three default ranges for player one against an $FCPA$ player two, in an $IR - 570$ card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to decrease this number.

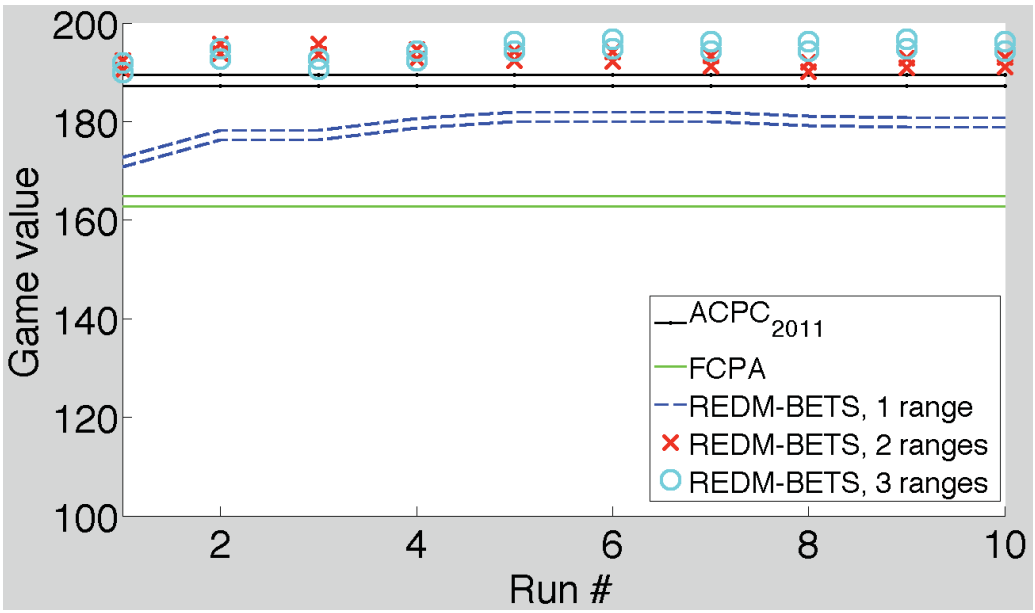


Figure 7.10: Bounds on game values of abstractions using one, two and three default ranges for player two against an $FCPA$ player one, in an $IR - 570$ card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number.

In comparing these results with $REDM$–$BETS$ applied to $IR - 100$ (Shown in Figures 7.3 through 7.8), the resemblance is striking. While some of the values have shifted a few milli-big blinds, the shapes of the curves are virtually identical. To get some insights in to why this is, I compare the most important bets in $IR - 100$ to those of $IR - 570$.

7.3.1 Bet size distributions

To see why the value curves of $IR - 100$ and $IR - 570$ are so similar, I now look at how the size and importance of bet sequences change as the distributions change. I took the top 25 bet-sizing players, as ranked by $R^T_{i,|\cdot|0}$, for the $IR - 100$ three range abstraction after run $r = 5$. The betting sequence of these bet sizing players, along with the bet size they use, is shown in the first and third columns of Table 7.3 and Table 7.4. The action sequences representing bet-sizing players can be interpreted as follows: for the main player the character 'd' represents the first range, 'e' represents the second range and 'g' represents the third range. For the opponent 'd' represents a pot sized bet. A '/' signifies the end of a round of betting. In each case the action sequence ends with the bet in question — for example the top three ranked bet-sizing players for player two (Table 7.4) in $IR - 100$ are 'e', 'd' and 'g' — these are ranges two, one and three, respectively, on the first action of the game.

Next I found the corresponding bet-sizing players in the $IR - 570$ three range abstraction after run $r = 5$. The fourth and fifth columns of the same tables show the relative importance of each of these bet-sizing players in the $IR - 570$ game, along with the bet size they use in that game.

Action sequence	$IR-100$ ranking	$IR-100$ bet size	$IR-570$ ranking	$IR-570$ bet size
dd	1	0.831	4	0.879
dg	2	2.547	3	2.457
de	3	0.808	2	0.891
cg	4	1.680	5	1.709
ce	5	1.539	6	1.455
cd	6	1.097	7	0.977
dc/d	7	0.191	1	0.190
ddc/d	8	0.210	8	0.236
dec/d	9	0.230	9	0.210
dc/cc/e	10	1.073	15	0.839
dc/cc/d	11	0.359	11	0.298
cgc/d	12	0.251	12	0.265
cdc/d	13	0.246	13	0.224
dgc/d	14	0.589	14	0.447
cec/d	15	0.276	10	0.226
dc/cdd	16	0.340	21	0.340
dc/cc/cc/d	17	0.409	24	0.230
dc/cc/g	18	1.408	16	1.210
cc/cc/d	19	0.510	26	0.510
dc/dc/d	20	0.250	18	0.240
ddc/dc/d	21	0.500	37	0.526
dc/e	22	0.610	17	0.600
dc/cdc/cc/d	23	0.287	34	0.190
dc/cc/ec/d	24	0.381	61	0.403
ddc/e	25	0.660	25	0.600

Table 7.3: Number of (I, a) pairs in various action abstractions for player one, generated using $REDM$–$BETS$ and $REFINED$–$BETS$. In each case $BETS$ was run five times. In all cases the opponent uses $FCPA$, and a 5 bucket perfect recall card abstraction of Texas hold'em was used.

Examining these tables, it's clear that while the order of importance of the bet-sizing players changes slightly, in general the same bet-sizing players are important in both cases. For player one (Table 7.3) there are only four bet-sizing players in the top 25 of $IR-100$ that are not in the top 25 of $IR-570$. The highest ranking of these bet-sizing players that didn't make the top 25 of $IR-570$ is bet-sizing player 19 "cc/cc/d", which ranks just one out of the top 25 for $IR-570$. Player two (Table 7.4) is similar — there are five bet-sizing players in the top 25 of $IR-100$ but not in the top 25 of $IR-570$, the highest ranking of which is bet-sizing player 16 "edc/cd".

Averaging the absolute difference in the bet sizes used by $IR-100$ and $IR-570$ for these bet-sizing players, I get an average absolute difference of 0.065 for player one and 0.061 for player two. The largest absolute difference in bet sizes is 0.198 for player one (row 18 of Table 7.3) and 0.251 for player two (row 11 of Table 7.4). It is clear from the similarity of the game value curves for each of these card abstractions that these differences in importance ranking the bet size have a very small impact on the game value. I explore this idea further in the next section.

Action sequence	$IR-100$ ranking	$IR-100$ bet size	$IR-570$ ranking	$IR-570$ bet size
e	1	0.830	2	0.843
d	2	0.724	1	0.794
g	3	1.210	3	1.210
ec/cd	4	0.393	5	0.441
dc/cd	5	0.400	4	0.397
gc/cd	6	0.440	8	0.439
dc/ce	7	0.600	6	0.600
ec/ce	8	0.600	7	0.600
ec/cdc/cd	9	0.493	15	0.710
cc/cd	10	0.510	10	0.510
dc/cdc/cd	11	0.506	16	0.757
ec/cc/cd	12	0.430	20	0.444
ec/cg	13	1.350	12	1.210
gc/ce	14	0.720	9	0.600
dc/cc/cd	15	0.440	19	0.400
edc/cd	16	0.495	32	0.580
ddc/cd	17	0.545	23	0.541
gc/cg	18	1.358	14	1.210
cc/ce	19	0.600	13	0.611
edd	20	0.994	46	0.973
dc/cg	21	1.420	11	1.210
edc/ce	22	0.600	22	0.600
dde	23	1.199	28	1.137
ec/cdc/ce	24	1.289	37	1.278
dc/cdc/ce	25	1.243	45	1.308

Table 7.4: Number of (I, a) pairs in various action abstractions for player two, generated using $REDM$–$BETS$ and $REFINED$–$BETS$. In each case $BETS$ was run five times. In all cases the opponent uses $FCPA$, and a 5 bucket perfect recall card abstraction of Texas hold'em was used.

7.4 Using action abstractions from $IR-100$ in $IR-570$

In the previous section I showed how similar the distribution of important bet sizes of $IR-100$ and $IR-570$ are on run $r=5$. In this section I go a step further and ask the question: if I took the action abstractions created in the $IR-100$ game, then paired them with the $IR-570$ card abstraction, what would the value be? The answer to this question is presented in Figures 7.11 and 7.12.

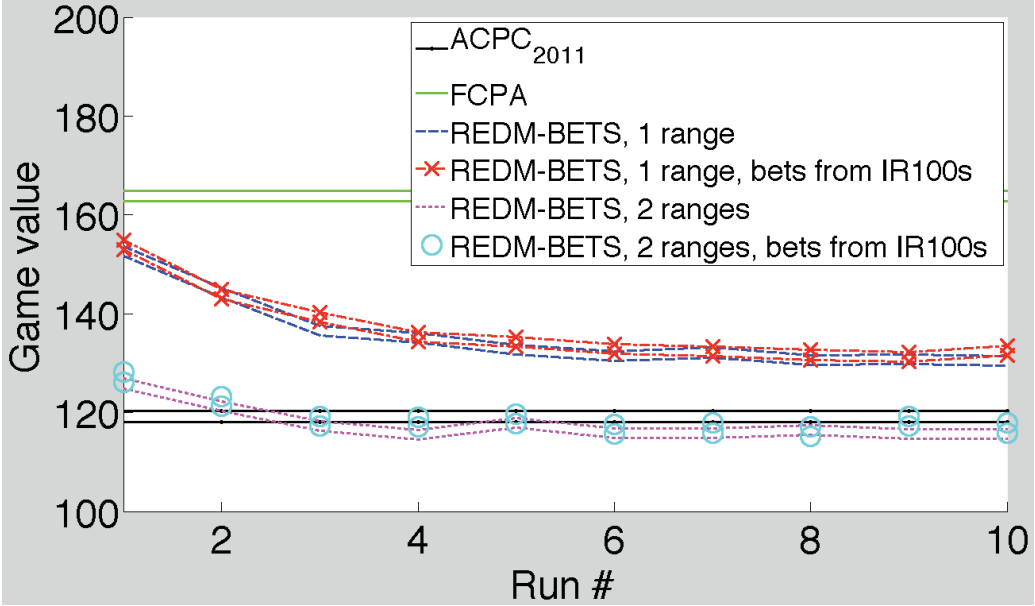


Figure 7.11: Bounds on game values of abstractions using one and two default ranges for player one against an $FCPA$ player two, in an $IR - 570$ card abstraction of Texas hold'em. Some bet sizes are calculated using an $IR - 100$ card abstraction before being paired with the $IR - 570$ card abstraction. The Y-axis is value to player two, so player one wishes to decrease this number.

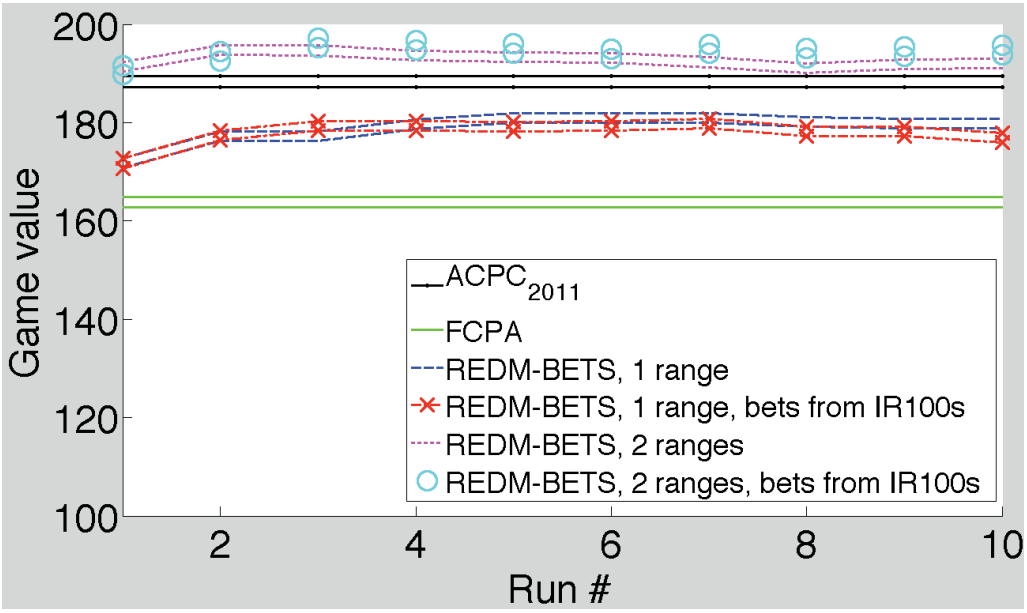


Figure 7.12: Bounds on game values of abstractions using one and two default ranges for player two against an $FCPA$ player one, in an $IR - 570$ card abstraction of Texas hold'em. Some bet sizes are calculated using an $IR - 100$ card abstraction before being paired with the $IR - 570$ card abstraction. The Y-axis is value to player two, so player two wishes to increase this number.

The game value curves of the action abstractions generated using an $IR - 100$ abstraction and

then applied to a $IR-570$ abstraction are virtually identical to those created in the $IR-570$ abstraction. For player one (Figure 7.11) there are a few points where the game value is slightly worse than that of the abstraction generated in $IR-570$, however player two actually shows the opposite effect effect as well — on a few runs the action abstraction from $IR-100$ actually performs slightly better in $IR-570$ than the abstraction created in $IR-570$! Overall, it is clear that there is little benefit to the game value if I use an $IR-570$ card abstraction to create my action abstractions, even if I eventually wish to pair my action abstractions with an $IR-570$ card abstraction. It is preferable to use a smaller card abstraction to create my action abstractions — larger card abstractions require more memory and a larger value of T for each run.[1]

Before continuing, recall that the game value curves in $IR-100$, while virtually identical in shape to those of $IR-570$, were shifted by a few milli-big blinds. To further illustrate this, I plot the game value curves from the $IR-100$ game and the $IR-570$ game on the same plot. The result is shown in Figures 7.13 and 7.14 for players one and two, respectively. Player one's value hits a worse peak value in the larger game, while the value curves for player two are very similar in both games. Combined, this indicates that the IR-570 game is slightly better for player two. I will return to this discussion in Section 7.6.

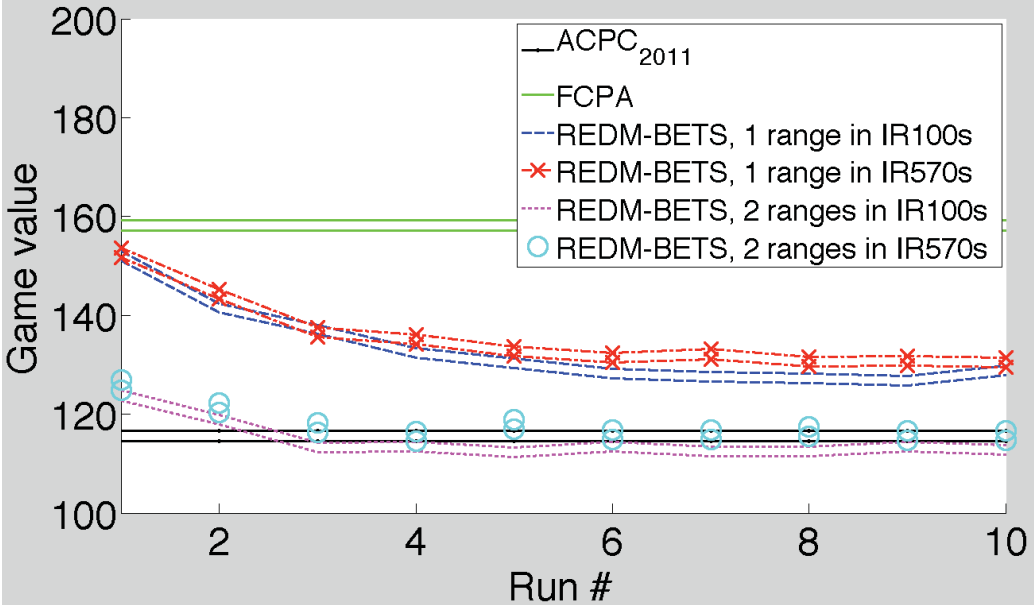


Figure 7.13: Bounds on game values of abstractions using one and two default ranges for player one against an $FCPA$ player two. The Y-axis is value to player two, so player one wishes to decrease this number.

[1]Note, however, that the time per iteration of $BETS$ does not increase as the number of buckets in each round is increased, as $BETS$ uses chance sampling, and thus it only considers one bucket on each round each iteration

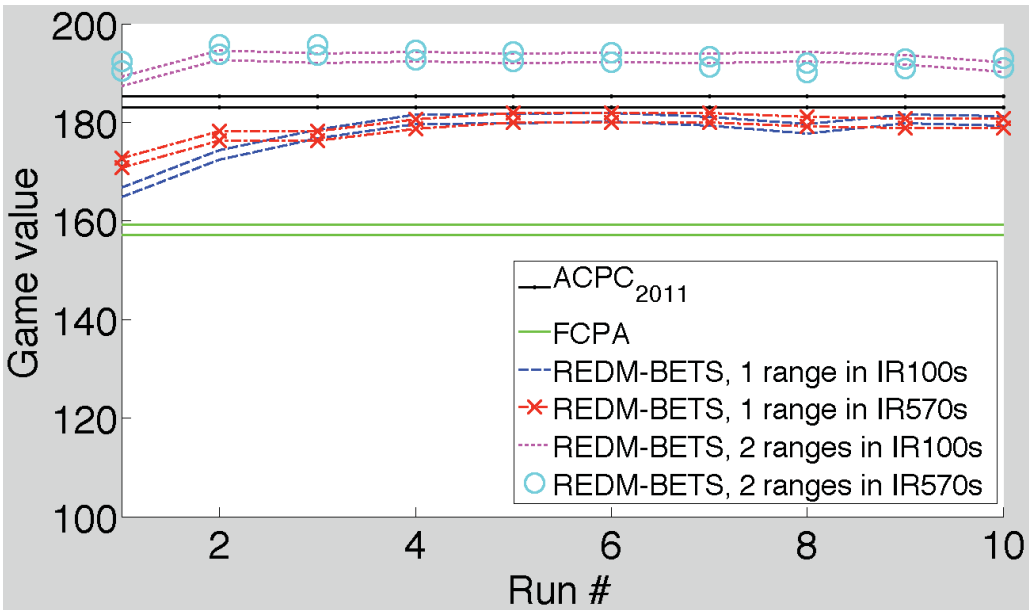


Figure 7.14: Bounds on game values of abstractions using one and two default ranges for player two against an $FCPA$ player one. The Y-axis is value to player two, so player two wishes to increase this number.

7.5 Larger opponent abstraction

Up to this point I have used an opponent action abstraction of $FCPA$ in all of my experiments. I now apply my algorithms to an imperfect recall $IR-100$ card abstraction paired with an $ACPC_{2011}$ opponent action abstraction. I start by examining the game value curve generated using one and two default ranges in this game. I used $T = 70,000,000$ iterations per run of $BETS$ for one default range and $T = 80,000,000$ for two default ranges. The results are shown in Figures 7.15 and 7.16 for players one and two, respectively.

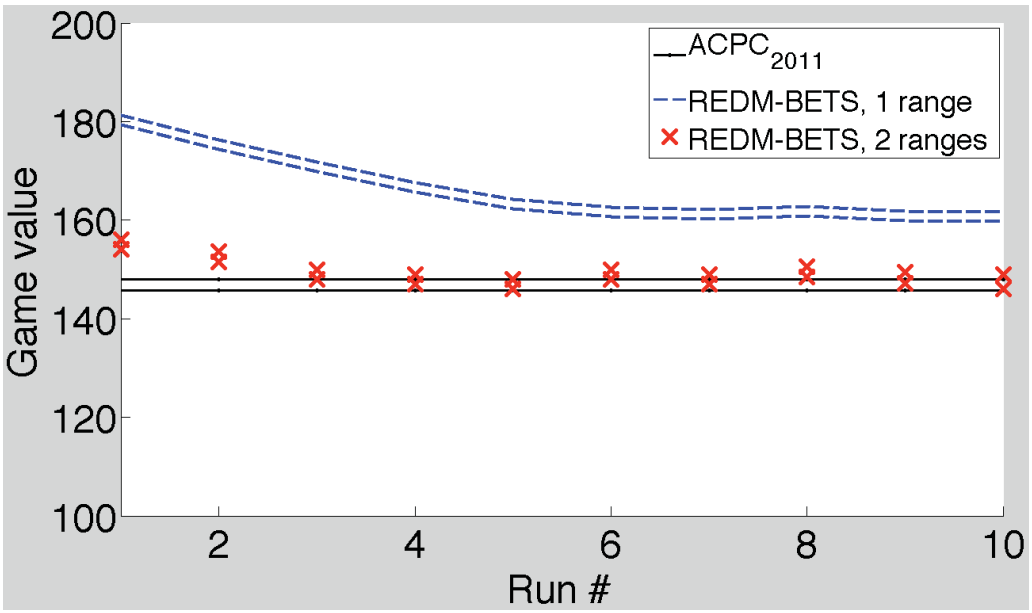


Figure 7.15: Bounds on game values of abstractions using one and two default ranges for player one against an $ACPC_{2011}$ player two, in an $IR - 100$ card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to decrease this number.

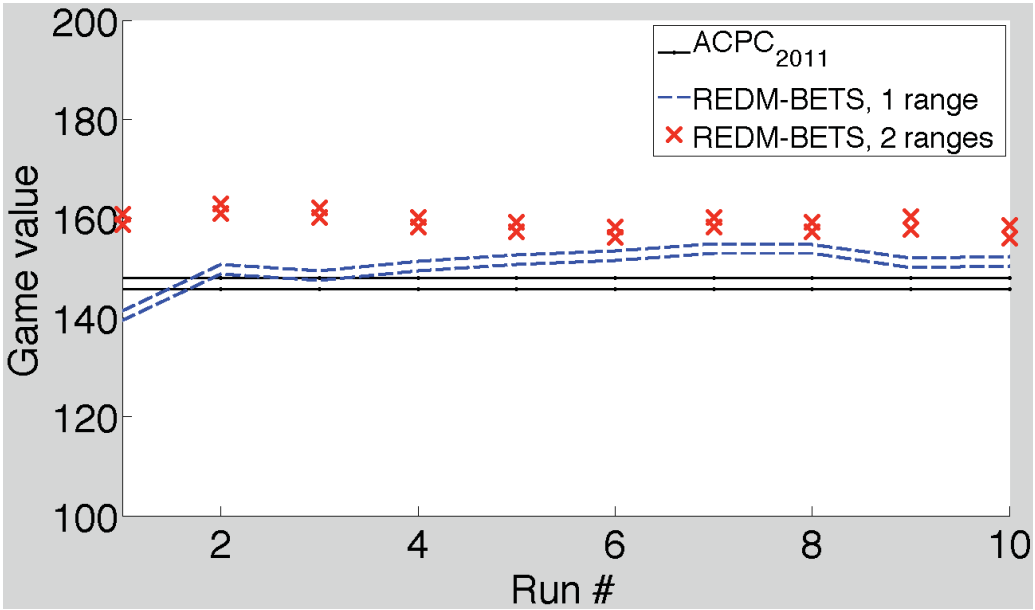


Figure 7.16: Bounds on game values of abstractions using one and two default ranges for player two against an $ACPC_{2011}$ player one, in an $IR - 100$ card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number.

Looking at these graphs, there is an interesting difference from all of the games with $FCPA$ opponents. In both $IR - 100$ and $IR - 570$, one range was never enough to beat the $ACPC_{2011}$

abstraction's value against $FCPA$. That remains the case here for player one, however for player two we can see that one range passes the value of $ACPC_{2011}$ played against itself. It is important to note, however, that while one range for player two beats the value of $ACPC_{2011}$ by a small margin, one range for player one falls short of the value of $ACPC_{2011}$ by a bigger margin. So combining the two values, one range still loses to $ACPC_{2011}$. For player one two default ranges improves upon the performance of one, roughly equally the value of $ACPC_{2011}$. For player two, again the value is improved over the one range peak value, though not by as much.

To apply my algorithms to three default ranges in this game proved to be very computationally demanding. Thus, instead of creating a full three range value curve, I ran the first iteration with three ranges for $T = 80,000,000$ iterations, and then pruned the tree. Pruning based on three ranges seemed a sensible approach, as in $IR-100$ against an $FCPA$ opponent using three default ranges led to greater peak game value for both players one and two (see Section 7.2.2). Multiple pruned versions of each player were created. Agents were then created for each of these pruned abstractions on run $r = 1$. $1,200,000,000$ hands of self-play were played to obtain the value of each pruned game before further runs of $BETS$. While two different abstractions having the same value after the first run does not guarantee they will peak at the same value, if adding bets leads to a higher value at this point, it is likely that the peak point will have a higher value as well.

The results of these self-play matches after pruning three ranges with various values of e on run $r = 1$ are shown in Tables 7.5 and 7.6 for players one and two, respectively. The number of (information set, action) pairs for each abstraction is also shown. Given the results of Section 7.2.2, I started with $e = 100$.

# extra bets (e)	Value to player two	# (I, a) pairs
100	148.57 ± 1.13	$20,115,544$
200	148.47 ± 1.11	$22,371,456$
500	147.69 ± 1.05	$25,692,726$

Table 7.5: Game values and number of (I, a) pairs in various action abstractions for player one, generated by completing just one run of $BETS$ using three ranges, then pruning with different values of e. In all cases player two uses $ACPC_{2011}$, and an $IR-100$ card abstraction of Texas hold'em was used. The game value to player two, so player one wishes to decrease this number.

# extra bets (e)	Value to player two	# (I, a) pairs
100	154.18 ± 1.08	$21,946,414$
200	157.75 ± 1.12	$23,263,498$
500	158.32 ± 1.15	$26,625,940$

Table 7.6: Game values and number of (I, a) pairs in various action abstractions for player two, generated by completing just one run of $BETS$ using three ranges, then pruning with different values of e. In all cases player one uses $ACPC_{2011}$, and an $IR-100$ card abstraction of Texas hold'em was used. The game value to player two, so player two wishes to increase this number.

Examining Table 7.5, note that $e = 100$ and $e = 200$ are very close in value — any gain from adding another 100 bets is so small that many more hands would have to be played to see

the difference. Moving up to $e = 500$ the game value moves downward (which is good, as it is value to player two), but only by 0.78 milli-big blinds. This gain is well within the 95% confidence intervals. Considering that $e = 500$ is 27.7% bigger than $e = 100$, I decided that this possible gain in value was not worth the increased abstraction size (and thus computational requirements) and choose $e = 100$ from this point on.

Applying the same analysis to Table 7.6, you can see there is a difference of 3.57 milli-big blinds between $e = 100$ and $e = 200$, well outside the confidence intervals. Increasing to $e = 500$ leads to an even smaller gain than it did for player one, only 0.57 milli-big blinds, again within the confidence intervals. I decided this was not worth the 14.5% increase in abstraction size and choose $e = 200$ going forward.

Given these results I applied $REFINED\text{–}BETS$ with $e = 100$ for player one and $e = 200$ for player two. The resulting game value curves, along with those of one and two ranges, are shown in Figures 7.17 and 7.18 for players one and two, respectively.

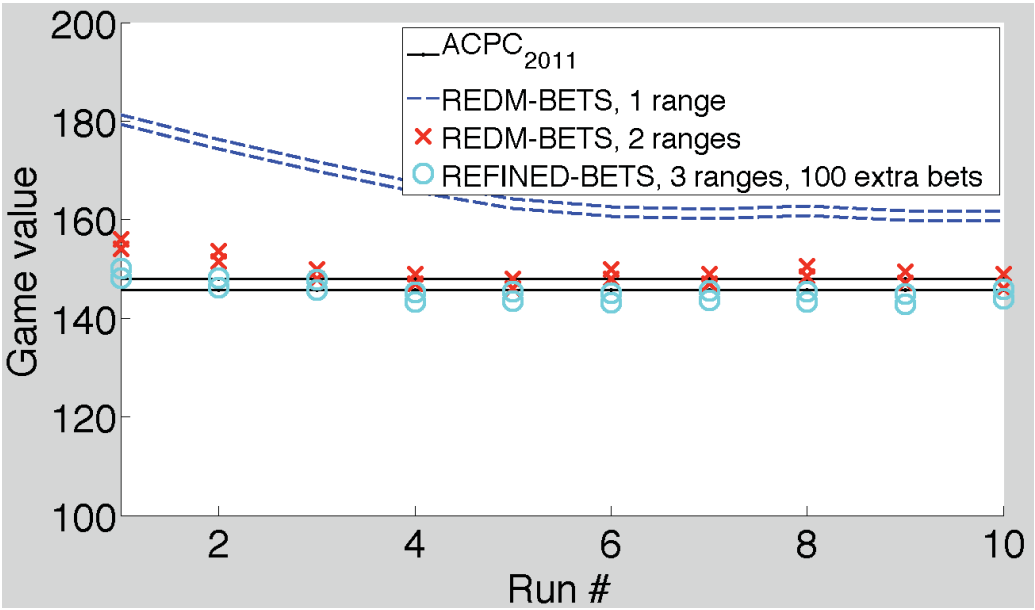


Figure 7.17: Bounds on game values of abstractions using one and two default ranges, along with three default range pruned with $e = 100$, for player one against an $ACPC_{2011}$ player two, in an $IR-100$ card abstraction of Texas hold'em. The Y-axis is value to player two, so player one wishes to decrease this number.

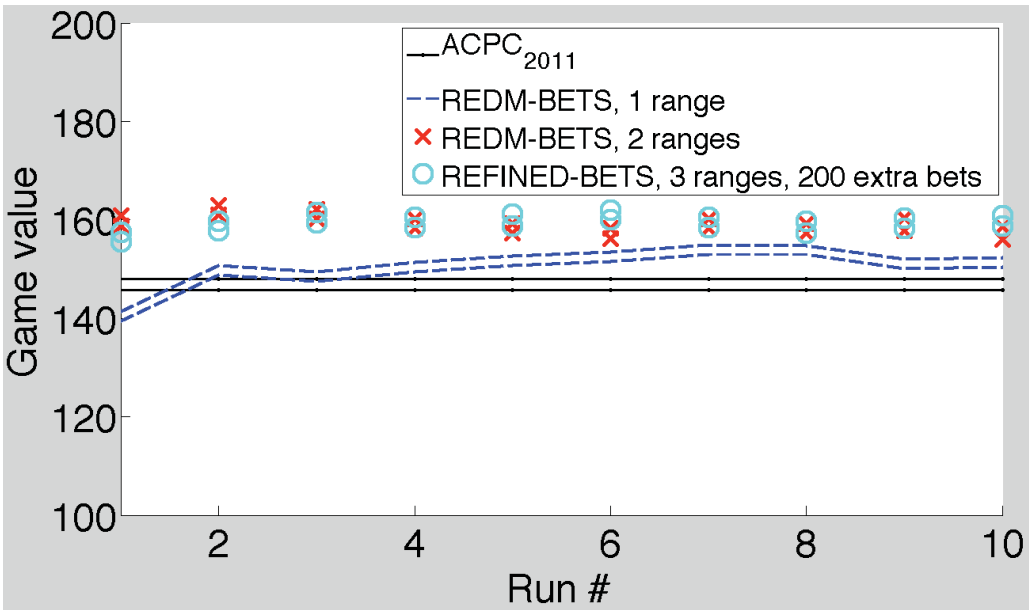


Figure 7.18: Bounds on game values of abstractions using one and two default ranges, along with three default range pruned with $e = 200$, for player two against an $ACPC_{2011}$ player one, in an $IR-100$ card abstraction of Texas hold'em. The Y-axis is value to player two, so player two wishes to increase this number.

As was the case with an $FCPA$ opponent, the two range curves and pruned three range curves have levelled off by run $r = 5$. Additionally, the values of the two range and the pruned three range curves are very similar. For player one (Figure 7.17), three pruned ranges performs better than two ranges all the way, peaking at a slightly higher value and beating $ACPC_{2011}$, while two ranges only ties it. For player two (Figure 7.18) two ranges peaks at roughly the same value as the three pruned ranges.

7.5.1 Size comparison

Now I compare the sizes of the two range abstractions to the sizes of the pruned three range abstractions. Instead of looking at run $r = 5$, as I have previously, I consider the sizes at various runs. The number of extra bets, along with the (information set, action) pairs, are shown in Tables 7.7 and 7.8.

Recall that the size of the abstractions created will change as bet sizes vary, as this can lead to the addition and removal of bet-sizing players (see Section 5.7). The purpose of this size comparison is to identify one abstraction for each player with a very good size-to-value ratio. In the next section I will explain how I then use these abstractions to create a no-limit Texas hold'em agent in a much larger card abstraction. The better the size-to-value ratio, the larger I can make the card abstraction.

Examining player one (Table 7.7) I note that the size of the two range abstractions reduces from runs $r = 2$ to $r = 4$, while in previous experiments the size of the abstractions created always increased or stayed constant after each run. While this is an interesting result, the reduction is quite

Run #	Abstraction	Value to player two	# (I, a) pairs
2	Start with three ranges, $e = 100$	147.14 ± 1.01	$21,714,330$
3	Start with three ranges, $e = 100$	146.78 ± 1.02	$23,895,330$
4	Start with three ranges, $e = 100$	144.29 ± 1.03	$24,192,102$
5	Start with three ranges, $e = 100$	144.49 ± 1.08	$24,526,474$
2	Two ranges, no pruning ($e = 3,567$)	152.57 ± 1.01	$27,366,116$
3	Two ranges, no pruning ($e = 3,616$)	149.02 ± 1.03	$27,334,530$
4	Two ranges, no pruning ($e = 3,670$)	148.06 ± 1.03	$26,008,944$
5	Two ranges, no pruning ($e = 3,722$)	146.98 ± 1.04	$26,499,513$
N/A	$ACPC_{2011}$ fixed bets	146.88 ± 1.09	$72,630,334$

Table 7.7: Number of (I, a) pairs in various action abstractions for player one, generated using $REDM$–$BETS$ and $REFINED$–$BETS$. The size of a symmetric fixed betting abstraction of $ACPC_{2011}$ is included for reference. In all cases the opponent uses $ACPC_{2011}$, and an $IR - 100$ card abstraction of Texas hold'em was used.

Run #	Abstraction	Value to player two	# (I, a) pairs
2	Start with three ranges, $e = 200$	158.78 ± 1.00	$26,781,540$
3	Start with three ranges, $e = 200$	160.45 ± 0.98	$28,135,768$
4	Start with three ranges, $e = 200$	159.51 ± 1.05	$31,725,582$
5	Start with three ranges, $e = 200$	160.02 ± 1.02	$34,146,196$
2	Two ranges, no pruning ($e = 4,168$)	161.93 ± 0.98	$38,519,400$
3	Two ranges, no pruning ($e = 4,728$)	161.23 ± 1.00	$44,841,628$
4	Two ranges, no pruning ($e = 5,236$)	159.31 ± 0.99	$51,622,628$
5	Two ranges, no pruning ($e = 5,811$)	158.24 ± 1.04	$56,321,020$
N/A	$ACPC_{2011}$ fixed bets	146.88 ± 1.09	$72,630,334$

Table 7.8: Number of (I, a) pairs in various action abstractions for player two, generated using $REDM$–$BETS$ and $REFINED$–$BETS$. The size of a symmetric fixed betting abstraction of $ACPC_{2011}$ is included for reference. In all cases the opponent uses $ACPC_{2011}$, and an $IR - 100$ card abstraction of Texas hold'em was used.

small, and on $r = 5$ the size increases again — overall the size of the two range abstractions seems to have stabilized over this time. More importantly, the pruned abstractions are smaller than the two range abstractions, if only by a small amount. As the sizes of these abstractions are all quite similar, I choose the highest value abstraction, on $r = 4$, to use with a larger card abstraction. Three ranges pruned at $r = 4$ is a minor $\approx 2\%$ improvement in game value over the fixed $ACPC_{2011}$ abstraction (see Figure 7.17), however it is only $\approx 33\%$ of the size.

The size of the player two abstractions show very different characteristics from the player one abstractions. The two range abstractions grow in size by $\approx 9 - 15\%$ each run. The pruned abstractions also grow in size, but at a slower pace, growing at $\approx 5\%$, $\approx 13\%$ and $\approx 8\%$ between runs 2, 3, 4 and 5 respectively. The two smallest peak abstractions here are $r = 3$ for the three range prune runs and $r = 2$ for the two range runs. The two range run peaks at a slightly higher value, however it is 37% larger than the three range pruned run at $r = 2$. Due to this large size difference for a very small value difference, I choose the three range pruned run on $r = 3$ to pair with a larger card abstraction. As with player one, this abstraction performs better than the fixed $ACPC_{2011}$ abstraction, this time by $\approx 9\%$ (see Figure 7.18), and it is only 39% of the size.

7.6 Scaling the card abstraction

I used the same procedure to create a poker agent in a much larger card abstraction as I did in Section 5.9 to create the winning entry of the 2012 ACPC bankroll instant run-off division. Recall that this involved taking the action abstraction for each player and pairing it with a much larger card abstraction, then applying the $MCCFR$ algorithm (the same implementation of this algorithm used in Section 5.9) [18, 40, 46]. I choose an $IR - 35000$ card abstraction — that is, an imperfect recall abstraction with 169 pre-flop buckets and $35,000$ buckets on each of the flop, turn and river. This card abstraction was generated using the same k-means technique that generated all other imperfect card recall abstractions used in this dissertation [42]. When paired with my action abstractions, the resulting abstractions required 126 gigabytes of RAM for player 1 and 146 gigabytes of RAM for player 2 when loaded by the MCCFR algorithm. MCCFR was run for approximately 68 days for player one and 66 days for player two. I used the same cluster as I did in Section 5.9, which has nodes of 4 AMD 6172 processors with 12 cores each, running at $2.1GHz$. Again, all 48 cores were used in the calculation. In total 309 billion iterations were run for player one, and 300 billion iterations for player two.

To see how this agent performs, I started by following my usual technique of playing $1,200,000,000$ hands of self-play to obtain tight empirical bounds on the game value. The values I obtained are 155.11 ± 1.14 and 162.54 ± 1.12 for players one and two, respectively. Subtracting player one's value from player two, this agent beats its opponent abstraction of $ACPC_{2011}$ by 7.43 ± 1.60. As shown in Figures 7.17 and 7.18, the corresponding values of this action abstraction in the card abstraction I used when training it, $IR - 100$, are144.29 ± 1.15 and 160.45 ± 1.12, again for players one and two respectively. Subtracting player one's value from player two we see that in $IR - 100$ this action abstraction beats its opponent abstraction of $ACPC_{2011}$ by 16.16 ± 1.61 in $IR - 100$.[2] I showed in Section 7.4 that while the value of abstractions trained in $IR - 100$ using an $FCPA$ opponent action abstraction shifted when applied to an $IR - 570$ game (with the same opponent action abstraction), the action abstractions created in the $IR - 570$ game had the same value. This suggests that the shift in game value may be influenced more by the card abstraction the action abstractions are eventually paired with, instead of the card abstractions used to create these action abstractions. With that in mind, it is hard to draw conclusions about whether or not action abstractions generated in the $IR - 35000$ game would significantly outperform these ones. What I can say is that my action abstractions continue to have a positive overall value against the opponent action abstraction, while being much smaller.

[2]The game values for player one and player two are sampled independently, so the combined confidence interval is calculated by adding the variances together

7.6.1 Results against top no-limit agents

Further testing of my no-limit Texas hold'em agent was done by playing it against other world-class Texas hold'em agents, all generated by the University of Alberta's Computer Poker Research Group (CPRG). As I did in Section 5.9, I used the translation system designed by Schnizlein et al. [58, 59], discussed in Section 3.8.2, to translate opponent bets not contained within the $ACPC_{2011}$ opponent action abstraction that I used. While in Section 5.9 I described how I used thresholding in the ACPC 2012 competition entry, no thresholding was used in this experiment. A cross table showing the result of all of these agents playing against one another is given in Table 7.9. Most agents in the cross table were generated using the same MCCFR algorithm to find an ϵ-Nash equilibrium, with the exception of $2011+$, which used the CFR algorithm. All of them used the same k-means clustering technique to create the card abstraction [42]. A key detailing the card and action abstraction of each agent in Table 7.9 is given in Table 7.10, along with a brief description of each agent. Agents $big2011$, $minBet_C$ and $minBet_B$ were all crafted to take up just under 250 gigabytes of memory in order to fit in 256 gigabytes of RAM.

	Opponents						
	$2011+$	$Hawkin_1$	$Hawkin_2$	$big2011$	$minBet_C$	$minBet_B$	Average
$2011+$		-26 ± 9	-57 ± 8	-35 ± 9	-47 ± 8	-53 ± 7	-42
$Hawkin_1$	26 ± 9		-45 ± 8	-23 ± 7	-27 ± 8	-27 ± 9	-20
$Hawkin_2$	57 ± 8	45 ± 8		14 ± 8	1 ± 7	2 ± 8	10
$big2011$	35 ± 9	23 ± 7	-14 ± 8		-7 ± 9	-2 ± 9	-3
$minBet_C$	47 ± 8	27 ± 8	-1 ± 7	7 ± 9		1 ± 8	5
$minBet_B$	53 ± 7	27 ± 9	-2 ± 8	2 ± 9	-1 ± 8		7
Max	57	45	15	22	26	16	

Table 7.9: One on one performance of top no-limit Texas hold'em agents in milli-big-blinds/game. Results were calculated by playing 40 duplicate matches of $100,000$ hands each ($8,000,000$ hands total). 95% confidence intervals on each result are provided.

Bot name	Action abstraction (in pot fractions)	Card Abstraction
$2011+$	$ACPC_{2011}$	$IR - 3700$
$Hawkin_1$	1 fixed-size range vs $ACPC_{2011}$ opponent	$IR - 18600$
$Hawkin_2$	3 variable-size ranges pruned vs $ACPC_{2011}$ opponent	$IR - 35000$
$big2011$	$ACPC_{2011}$ + 0.5 once pre-flop, 1.5, 6, 20, 40	$IR - 9000$ flop and turn, $IR - 3700$ river
$minBet_C$	$ACPC_{2011}$ + 1 minimum bet per round	$IR - 9000$ flop and turn, $IR - 1175$ river
$minBet_B$	$ACPC_{2011}$ + 2 + 1 minimum bet per round	$IR - 3700$ flop and turn, $IR - 1175$ river

Table 7.10: Abstractions used by the agents in Table 7.9

$Hawkin_2$, the agent described in Section 7.6, performs the best of all agents in the cross table,

Bot name	Description
2011+	Upgraded version of the winner of the $ACPC$ instant run-off division in 2011. The original agent was created using $7,600,000$ iterations of CFR, this one was run for $45,000,000$ iterations.
$Hawkin_1$	Bot described in Section 5.9. Winner of the 2012 ACPC instant run-off division.
$Hawkin_2$	Bot described in Section 7.6.
$big2011$	Bot created by expanding 2011+ in both cards and actions.
$minBet_C$ $minBet_B$	Two agents containing a maximum of one minimum bet per round. $minBet_C$ had more buckets in the card abstraction, while $minBet_B$ had an extra bet (2 pot) in the betting abstraction.

Table 7.11: Descriptions of the agents in Table 7.9

averaging a win of 10 milli-big-blinds per game. It easily beats the winners of the 2011 and 2012 instant run-off competitions, winning over these agents by the largest margin of any agent in the cross table. The victory over the upgraded 2011 winner makes a great deal of sense, as the bet sizes used by that agent were the ones I used as the opponent action abstraction. As my agent wins in self-play against the fixed bets of $ACPC_{2011}$ by 7.43 ± 1.60 (see Section 7.6), the remainder of the value comes from the combination of the fact that my agent has a much larger card abstraction ($IR - 35000$ vs $IR - 3700$), and the advantage of causing translation problems for this agent that only understands $ACPC_{2011}$ bets. The other of these agents, the 2012 winner, was generated using $RE\text{–}BETS$ with one fixed-width default range, as described in Section 5.9. This agent has closer to the same card abstraction, using $IR - 18600$. The advantage of translation is nullified here, as both agents use $ACPC_{2011}$ as their opponent action abstraction, but bet with their own unique distributions of bets.

While $Hawkin_2$ defeats $big2011$ with statistical significance, agents $minBet_C$ and $minBet_B$ have similar performance to $Hawkin_2$ overall, and the matches between these bots do not have a clear victor — for this many more hands would be required. As $Hawkin_2$ makes many bets that are not within the opponent abstraction of the other bots, they must use translation when interpreting these bets. Similarly, as minimum bets are not within the opponent abstraction used by $Hawkin_2$ (the $ACPC_{2011}$ action abstraction), it must translate these bets to a bet size it understand before it can respond to them. If we expect to be up against an agent that uses minimum bets, then we can include them in the opponent action abstraction, and this should significantly improve our performance. Note that the opposite option is not available to the opponent agents - as $Hawkin_2$ makes a different bet size at each point in the tree, unless the opposing agent-maker has the entire list of bet sizes available to them when creating their agent, they cannot avoid translating the bet sizes at game-time.

Chapter 8

Conclusion

I conclude this dissertation with a summary of the contributions of this work and the limitations of my approach.

8.1 Contributions

I have introduced a set of algorithms for the automated creation of action abstractions in games with large or continuous action spaces such as no-limit hold'em. I started in Chapter 1 with a summary of the available poker literature pertaining to bet sizing, as well as a discussion of the computational difficulty of no-limit poker games. In Chapter 2 I gave a detailed description of all no-limit poker games studied in this dissertation. Chapter 3 was devoted to relevant game theory background, as well as a summary of related work. In Chapters 4 through 7 I introduced a game transformation and multiple accompanying regret-minimizing algorithms designed to automatically abstract large or continuous action spaces, and I showed the successful application of these algorithms to various no-limit poker games, most notably no-limit Texas hold'em.

8.1.1 Game transformation and $BETS$

Abstraction of the action space is essential in games with large or continuous action spaces, such as no-limit Texas hold'em. In Chapter 4 I introduced a transformation that can be applied to such games. This transformation creates a multiplayer game, and the strategies of the added players in this transformed game can be mapped to an action abstraction of the original game. I introduced the $BETS$ algorithm, a regret minimizing algorithm designed to find an ϵ-Nash equilibrium of the transformed game. I showed that in a number of small poker games, this ϵ-Nash equilibrium can then be mapped to an action abstraction of the original game that uses (approximately) the Nash equilibrium bet sizes.

8.1.2 RE–$BETS$

Chapter 5 explained some of the shortcomings of $BETS$ in larger games. Bounding the range of possible actions is necessary for my game transformation, however using one range for all parameter-setting agents does not scale well with game size. To remedy this problem I introduced RE–$BETS$, an algorithm that iteratively applies $BETS$, adjusting the ranges of various parameter-setting agents between each iteration. RE–$BETS$ shows improved performance in small games, and scales well with game size. Using RE–$BETS$ I created a no-limit Texas hold'em poker agent that won the 2011 $ACPC$ no-limit bankroll instant run-off competition.

8.1.3 $REDM$–$BETS$ and $REFINED$–$BETS$

In Chapter 6 I improved upon the range-adjusting done on each iteration of RE–$BETS$. While RE–$BETS$ keeps the width of the ranges of parameter-setting agents constant, I showed that instead it is advantageous to keep the pot growth ratio of the upper and lower bound constant. This led to the RED–$BETS$ algorithm. I then generalized this algorithm, creating the $REDM$–$BETS$ algorithm, which allows for multiple parameter-setting agents at each decision node. While using multiple parameter setting agents can improve performance, I showed that there is value in keeping multiple agents only at key points in the game. This is accomplished using the $REFINED$–$BETS$ algorithm. When applied to a perfect recall card abstraction of no-limit Texas hold'em, $REFINED$–$BETS$ creates action abstractions that are smaller than those created by $REDM$–$BETS$ while achieving the same game value.

8.1.4 Changing the card abstraction

In Chapter 7 I studied the application of $REDM$–$BETS$ and $REFINED$–$BETS$ to different card and action abstractions. When using imperfect recall card abstractions $REFINED$–$BETS$ outperforms $REDM$–$BETS$, but by a smaller margin than in perfect recall games. Additionally, the action abstractions these algorithms create when applied to an imperfect recall card abstraction do not change significantly as the size of the imperfect recall card abstraction is increased. This leads to the idea of using a small imperfect recall card abstraction when generating an action abstraction, and then pairing the resulting action abstraction with a much larger card abstraction to create the final no-limit Texas hold'em agent. I used $REFINED$–$BETS$ and this technique to generate an agent to play no-limit Texas hold'em, and I showed that it performs very well against a collection of world-class no-limit Texas hold'em poker agents.

8.2 Limitations of this work

8.2.1 Separate runs to create action abstractions and produce strategies

The output of the $BETS$ algorithm, along with all of the other algorithms presented in this thesis, is an action abstraction. One must apply the algorithm, extract the action abstraction, and then run another regret-minimizing algorithm such as CFR using this action abstraction to create an agent that can play the original game. I believe that it may be possible to combine these steps, which is an excellent topic for future work.

8.2.2 Serial computation of $BETS$

The $BETS$ algorithm has not, to this point, been implemented in parallel. This restricts the size of games to which it can be applied. While the action abstractions I create scale well with increasing card abstractions, parallelization would allow usage of more complex opponent action abstractions.

8.2.3 Two strategy calculations for one agent

In most of the experiments in this dissertation, the $BETS$ algorithm is applied to only one player, while the other player uses a hand picked action abstraction (I refer to this throughout as the opponent action abstraction). This is done because there are certain pot fractions that are used more often than others by both poker professionals and poker agents. For this reason the algorithms are applied twice, once for each player. A poker agent is then obtained by using the results of one calculation when playing as one of the players, and the other calculation when playing as the other player. Thus obtaining a strategy to play using the action abstractions I generate requires double the computation it would if both players bet using the same pot fractions. This limitation is offset by the fact that my action abstractions are quite small, and thus it takes less time to compute each strategy.

8.2.4 Local maxima

In Section 4.5 I showed that the $BETS$ algorithm can get stuck in local maxima. This is possible because my transformation creates a multiplayer game, and thus there can be multiple Nash equilibria with different values. I revisited this issue in Section 6.8 and discussed how the $REFINED$–$BETS$ algorithm helps in avoiding local maxima. I showed that, in half-street Kuhn poker, $REFINED$–$BETS$ produces the global maxima as long as one of the bet-sizing agents being used has a range such that $0 \leq L < H \leq 1$. Furthermore, in Section 6.2 I showed that multiple different initial conditions for RE–$BETS$ lead to action abstractions with approximately the same game value. This is a potential indication that, even if there are multiple local maxima with different values, RE–$BETS$ either finds the same ones or the values of these maxima are very similar. It is by no means a proof of this, and it is quite possible that there are global maxima with much greater values that my algorithms are not finding.

8.2.5 Lack of a convergence proof for $BETS$

In Chapter 4 I show that the $BETS$ algorithm converges to Nash equilibrium bet sizes for multiple small no-limit poker games. I do not, however, have a proof of convergence for the algorithm, nor have I proven that if the algorithm converges, it has converged to an ϵ-Nash equilibrium of the transformed game. There may be certain conditions that are required for this to occur. What I can say is that I have not seen any examples of cases where the algorithm failed to converge.

8.2.6 Choice of variable N in $BETS$

When I introduced $BETS$ in Section 4.3, I used the variable N, which determines the rate at which bet-sizing agents can change their effective bet size from iteration to iteration. After some initial investigation in half-street Kuhn poker, I set $N = 10,000$, and used that value for all of the experiments in the remainder of the dissertation. It may be the the speed with which the algorithm converges is directly linked to the setting of this variable. The optimal value of N may be different for different games, or even different for each parameter-setting agent in the same game.

8.3 Final thoughts

I have shown that there is much value to be gained by automating the abstraction of the action space in no-limit poker games. Simply adding more bet sizes to the action abstraction of both players and solving larger and larger games is not a scalable approach. I have outlined algorithms that are very scalable, both from a computation and storage standpoint. My experimental results show that the resulting abstractions have greater value than larger symmetrical abstractions and take up less memory. Additionally, as was discussed at the end of Chapter 7, the unpredictable nature of the bet sizes my abstractions use has added advantages against computer opponents. Most approaches to creating poker agents require a predetermined opponent action abstraction, and thus opponents of agents created using the algorithms outlined in this dissertation will be required to use translation to interpret these unusual bet sizes.

In summary, the research described in this dissertation has made the following contributions:

- I introduced a transformation and accompanying regret-minimizing algorithm ($BETS$) that is shown experimentally to generate high value action abstractions (Chapter 4).
- I introduced an algorithm that iteratively applies $BETS$, known as RE–$BETS$, which resolves some of the issues with scaling the $BETS$ algorithm to large games (Chapter 5).
- I further refined the RE–$BETS$ algorithm to allow for multiple parameter-setting agents at a single decision node. This led to the $REDM$–$BETS$ algorithm (Chapter 6).
- I introduced a metric that can be used to rank the importance of parameter-setting agents. This is used by my final algorithm, $REFINED$–$BETS$, to prune the game tree (Chapter 6).

- I showed that these algorithms scale well – that when applying these algorithms to small imperfect recall poker games, the resulting action abstractions perform well when paired with larger card abstractions (Chapter 7).
- I applied $RE\text{–}BETS$ and $REFINED\text{–}BETS$ to create state-of-the-art no-limit poker agents (Chapter 5, Chapter 7).

Appendix A

Game value tables

Here I present tables showing the game values for all game value curves presented in Chapters 5, 6 and 7.

Run #	RE-BETS
1	51.14 ± 2.4
2	42.8 ± 2.59
3	36.04 ± 2.98
4	32.25 ± 3.32
5	30.54 ± 3.26
6	31.51 ± 3.31
7	30.68 ± 3.3
8	29.48 ± 3.32
9	30.36 ± 3.52
10	30.36 ± 3.51

Table A.1: Value to player two and bounds for the curve shown in Figure 5.4

Run #	RE-BETS
1	77.88 ± 2.49
2	92.98 ± 1.41
3	101.84 ± 1.85
4	107.82 ± 1.68
5	110.99 ± 1.81
6	115.79 ± 1.87
7	115.38 ± 1.82
8	113.07 ± 2.11
9	111.64 ± 2.8
10	111.74 ± 2.8

Table A.2: Value to player two and bounds for the curve shown in Figure 5.5

Run #	RE-BETS [0.4, 0.6]	RE-BETS [0.8, 1.0]	RE-BETS [1.2, 1.4]
1	39.6 ± 5.6	51.8 ± 2.2	60.84 ± 2.39
2	34.75 ± 5.75	43.35 ± 2.65	56.08 ± 2.74
3	30.25 ± 6.25	36.5 ± 2.9	51.16 ± 3.05
4	29.75 ± 6.25	30.25 ± 3.25	47.02 ± 2.73
5	29.69 ± 6.26	29.47 ± 3.15	42.97 ± 2.17
6	29.22 ± 6.41	28.73 ± 3.2	39.42 ± 2.58
7	29.73 ± 6.07	28.32 ± 3.15	36.08 ± 2.67
8	29.82 ± 5.71	27.6 ± 3.56	32.59 ± 2.8
9	29.12 ± 4.94	27.83 ± 3.73	30.16 ± 2.73
10	29.38 ± 4.84	28.16 ± 3.57	29.43 ± 3.15

Table A.3: Value to player two and bounds for each curve shown in Figure 6.1

Run #	RE-BETS [0.4, 0.6]	RE-BETS [0.8, 1.0]	RE-BETS [1.2, 1.4]
1	104.88 ± 2.81	85.5 ± 2	45.88 ± 2.78
2	110.58 ± 2.98	95.25 ± 1.75	54.31 ± 2.76
3	113 ± 2.65	102 ± 1.5	61.39 ± 3
4	114.01 ± 3.12	108.15 ± 1.65	67.02 ± 2.86
5	112.77 ± 2.67	113.6 ± 1.73	76.15 ± 2.43
6	113.35 ± 2.79	115.81 ± 1.69	90.87 ± 1.64
7	113.36 ± 2.86	115.89 ± 1.87	102.46 ± 1.55
8	111.18 ± 3.23	116.99 ± 1.62	109.11 ± 1.87
9	111.96 ± 3.24	116.85 ± 1.89	115.63 ± 1.71
10	110.32 ± 3.14	117.78 ± 1.99	117.06 ± 1.54

Table A.4: Value to player two and bounds for each curve shown in Figure 6.2

Run #	RED-BETS [0.71, 0.94]	RE-BETS [0.8, 1.0]
1	47.26 ± 2.18	51.8 ± 2.2
2	36.55 ± 2.44	43.35 ± 2.65
3	31 ± 2.24	36.5 ± 2.9
4	28.83 ± 2	30.25 ± 3.25
5	28.44 ± 1.9	29.47 ± 3.15
6	28.09 ± 1.9	28.73 ± 3.2
7	27.58 ± 1.93	28.32 ± 3.15
8	27.07 ± 1.96	27.6 ± 3.56
9	26.74 ± 1.97	27.83 ± 3.73
10	25.8 ± 1.97	28.16 ± 3.57

Table A.5: Value to player two and bounds for each curve shown in Figure 6.3

Run #	RED-BETS [0.71, 0.94]	RE-BETS [0.8, 1.0]
1	93.81 ± 1.89	85.5 ± 2
2	103.96 ± 1.35	95.25 ± 1.75
3	108 ± 1.38	102 ± 1.5
4	114.91 ± 1.28	108.15 ± 1.65
5	118.16 ± 1.31	113.6 ± 1.73
6	118.43 ± 1.26	115.81 ± 1.69
7	118.23 ± 1.18	115.89 ± 1.87
8	118.44 ± 1.2	116.99 ± 1.62
9	117.02 ± 1.29	116.85 ± 1.89
10	118.15 ± 1.22	117.78 ± 1.99

Table A.6: Value to player two and bounds for each curve shown in Figure 6.4

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges
1	47.26 ± 2.18	27.22 ± 2.41	26.25 ± 2.44
2	36.55 ± 2.44	25.13 ± 2.34	24.31 ± 2.41
3	31 ± 2.24	24.51 ± 2.3	23.44 ± 2.24
4	28.83 ± 2	24.15 ± 2.29	23.28 ± 2.52
5	28.44 ± 1.9	23.19 ± 2.23	22.23 ± 2.68
6	28.09 ± 1.9	21.96 ± 2.16	21.03 ± 2.6
7	27.58 ± 1.93	20.78 ± 2.23	19.72 ± 2.45
8	27.07 ± 1.96	20.92 ± 2.14	19.6 ± 2.36
9	26.74 ± 1.97	21.19 ± 2.15	19.89 ± 2.41
10	25.8 ± 1.97	20.34 ± 2.25	19.69 ± 2.44

Table A.7: Value to player two and bounds for each curve shown in Figure 6.5

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges
1	93.81 ± 1.89	116.33 ± 2.41	116.69 ± 2.56
2	103.96 ± 1.35	122.51 ± 2.34	123.16 ± 2.49
3	108 ± 1.38	122.72 ± 2.15	125.02 ± 2.53
4	114.91 ± 1.28	122.99 ± 2.11	125.08 ± 2.63
5	118.16 ± 1.31	122.73 ± 2	124.96 ± 2.68
6	118.43 ± 1.26	123.35 ± 1.83	124.88 ± 2.66
7	118.23 ± 1.18	122.95 ± 1.82	124.66 ± 2.41
8	118.44 ± 1.2	123.91 ± 2.03	124.92 ± 2.59
9	117.02 ± 1.29	123.4 ± 1.77	124.93 ± 2.61
10	118.15 ± 1.22	124.38 ± 2.19	124.91 ± 2.26

Table A.8: Value to player two and bounds for each curve shown in Figure 6.6

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REFINED-BETS 2 Ranges 1 Extra Bet	REFINED-BETS 2 Ranges 25 Extra Bets
1	47.26 ± 2.18	27.22 ± 2.41	29.05 ± 2.25	27.24 ± 2.28
2	36.55 ± 2.44	25.13 ± 2.34	27.66 ± 2.11	25.22 ± 2.12
3	31 ± 2.24	24.51 ± 2.3	26.75 ± 2.09	24.45 ± 2.12
4	28.83 ± 2	24.15 ± 2.29	26.11 ± 2.02	23.65 ± 2.05
5	28.44 ± 1.9	23.19 ± 2.23	25.49 ± 1.96	22.86 ± 2.13
6	28.09 ± 1.9	21.96 ± 2.16	24.18 ± 1.91	22.09 ± 2.11
7	27.58 ± 1.93	20.78 ± 2.23	23.48 ± 1.95	21.26 ± 2.12
8	27.07 ± 1.96	20.92 ± 2.14	23.16 ± 1.93	20.73 ± 2.1
9	26.74 ± 1.97	21.19 ± 2.15	23.25 ± 1.99	21.28 ± 2.21
10	25.8 ± 1.97	20.34 ± 2.25	23.49 ± 2.03	21.36 ± 2.19

Table A.9: Value to player two and bounds for each curve shown in Figure 6.7

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REFINED-BETS 2 Ranges 1 Extra Bet	REFINED-BETS 2 Ranges 25 Extra Bets
1	93.81 ± 1.89	116.33 ± 2.41	110.28 ± 1.98	116.05 ± 2.29
2	103.96 ± 1.35	122.51 ± 2.34	117.24 ± 1.53	121.77 ± 2.05
3	108 ± 1.38	122.72 ± 2.15	118.26 ± 1.61	122.62 ± 1.96
4	114.91 ± 1.28	122.99 ± 2.11	118.9 ± 1.46	123.24 ± 2.02
5	118.16 ± 1.31	122.73 ± 2	118.79 ± 1.44	122.61 ± 1.87
6	118.43 ± 1.26	123.35 ± 1.83	118.74 ± 1.48	122.61 ± 1.96
7	118.23 ± 1.18	122.95 ± 1.82	118.99 ± 1.44	123.06 ± 1.98
8	118.44 ± 1.2	123.91 ± 2.03	118.53 ± 1.3	123.08 ± 1.98
9	117.02 ± 1.29	123.4 ± 1.77	118.39 ± 1.5	123.13 ± 2.03
10	118.15 ± 1.22	124.38 ± 2.19	118.25 ± 1.51	122.73 ± 1.9

Table A.10: Value to player two and bounds for each curve shown in Figure 6.8

Run #	REDM-BETS 1 Range Statistical bounds	REDM-BETS 1 Range Absolute bounds
1	45.73 ± 0.98	47.26 ± 2.18
2	36.05 ± 0.98	36.55 ± 2.44
3	31.63 ± 0.98	31 ± 2.24
4	28.5 ± 0.98	28.83 ± 2
5	28.04 ± 0.98	28.44 ± 1.9
6	26.2 ± 0.98	28.09 ± 1.9
7	27.16 ± 0.98	27.58 ± 1.93
8	26.08 ± 0.98	27.07 ± 1.96
9	24.61 ± 0.98	26.74 ± 1.97
10	27.21 ± 0.98	25.8 ± 1.97

Table A.11: Value to player two and bounds for each curve shown in Figure 7.1

Run #	REDM-BETS 1 Range Statistical bounds	REDM-BETS 1 Range Absolute bounds
1	95.53 ± 0.98	93.81 ± 1.89
2	103.64 ± 0.98	103.96 ± 1.35
3	108.33 ± 0.98	108 ± 1.38
4	115.56 ± 0.98	114.91 ± 1.28
5	117.61 ± 0.98	118.16 ± 1.31
6	117.92 ± 0.98	118.43 ± 1.26
7	119.12 ± 0.98	118.23 ± 1.18
8	117.22 ± 0.98	118.44 ± 1.2
9	117.39 ± 0.98	117.02 ± 1.29
10	117.08 ± 0.98	118.15 ± 1.22

Table A.12: Value to player two and bounds for each curve shown in Figure 7.2

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges
1	152.02 ± 0.94	123.78 ± 0.98	115.52 ± 0.96
2	141.47 ± 0.93	118.94 ± 0.98	113.18 ± 0.96
3	137.12 ± 0.94	113.21 ± 0.98	111 ± 0.96
4	132.34 ± 0.95	113.45 ± 0.97	109.79 ± 0.96
5	130.24 ± 0.97	112.34 ± 0.97	107.65 ± 0.95
6	128.27 ± 0.98	113.46 ± 0.97	106.97 ± 0.98
7	127.63 ± 0.97	112.39 ± 0.97	108.53 ± 0.98
8	127.25 ± 0.97	112.42 ± 0.96	108.06 ± 0.98
9	126.7 ± 0.98	113.35 ± 0.96	108.39 ± 0.98
10	128.9 ± 0.98	112.71 ± 1.01	108.03 ± 1

Table A.13: Value to player two and bounds for each curve shown in Figure 7.3

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges
1	165.78 ± 0.94	188.28 ± 0.98	189.05 ± 0.93
2	173.36 ± 0.94	193.67 ± 0.98	192.44 ± 0.94
3	177.63 ± 0.94	192.96 ± 0.98	194.54 ± 0.94
4	180.58 ± 0.94	193.3 ± 0.98	195.06 ± 0.94
5	180.76 ± 0.94	193.06 ± 0.98	195.47 ± 0.94
6	181.04 ± 0.94	193.17 ± 0.98	195.54 ± 0.94
7	180.22 ± 0.95	192.98 ± 0.98	195.6 ± 0.94
8	178.69 ± 0.94	193.39 ± 0.98	194.53 ± 0.94
9	180.72 ± 0.95	192.63 ± 0.98	194.54 ± 0.94
10	180.32 ± 0.95	191.29 ± 0.98	193.6 ± 0.94

Table A.14: Value to player two and bounds for each curve shown in Figure 7.4

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges	REFINED -BETS 3 Ranges 1 Extra Bet	REFINED -BETS 3 Ranges 25 Extra Bets
1	152.02 ± 0.94	123.78 ± 0.98	115.52 ± 0.96	129.2 ± 0.98	118.47 ± 0.98
2	141.47 ± 0.93	118.94 ± 0.98	113.18 ± 0.96	121.93 ± 0.98	116.74 ± 0.98
3	137.12 ± 0.94	113.21 ± 0.98	111 ± 0.96	119.51 ± 0.98	114.26 ± 0.98
4	132.34 ± 0.95	113.45 ± 0.97	109.79 ± 0.96	119.54 ± 0.98	114.78 ± 0.98
5	130.24 ± 0.97	112.34 ± 0.97	107.65 ± 0.95	119.89 ± 0.98	112.38 ± 0.98
6	128.27 ± 0.98	113.46 ± 0.97	106.97 ± 0.98	119.53 ± 0.98	112.55 ± 0.98
7	127.63 ± 0.97	112.39 ± 0.97	108.53 ± 0.98	121.58 ± 0.98	114.33 ± 0.98
8	127.25 ± 0.97	112.42 ± 0.96	108.06 ± 0.98	121.55 ± 0.98	114.76 ± 0.98
9	126.7 ± 0.98	113.35 ± 0.96	108.39 ± 0.98	121.74 ± 0.98	116.34 ± 0.98
10	128.9 ± 0.98	112.71 ± 1.01	108.03 ± 1	121.16 ± 0.98	115.33 ± 0.98

Table A.15: Value to player two and bounds for each curve shown in Figure 7.5

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges	REFINED -BETS 3 Ranges 1 Extra Bet	REFINED -BETS 3 Ranges 25 Extra Bets
1	165.78 ± 0.94	188.28 ± 0.98	189.05 ± 0.93	172.77 ± 0.98	186.3 ± 0.98
2	173.36 ± 0.94	193.67 ± 0.98	192.44 ± 0.94	176.93 ± 0.98	188.33 ± 0.98
3	177.63 ± 0.94	192.96 ± 0.98	194.54 ± 0.94	180.15 ± 0.98	188.55 ± 0.98
4	180.58 ± 0.94	193.3 ± 0.98	195.06 ± 0.94	182.94 ± 0.98	188.98 ± 0.98
5	180.76 ± 0.94	193.06 ± 0.98	195.47 ± 0.94	184.46 ± 0.98	190.66 ± 0.98
6	181.04 ± 0.94	193.17 ± 0.98	195.54 ± 0.94	185.79 ± 0.98	189.62 ± 0.98
7	180.22 ± 0.95	192.98 ± 0.98	195.6 ± 0.94	184.46 ± 0.98	188.27 ± 0.98
8	178.69 ± 0.94	193.39 ± 0.98	194.53 ± 0.94	185.29 ± 0.98	188.29 ± 0.98
9	180.72 ± 0.95	192.63 ± 0.98	194.54 ± 0.94	185.69 ± 0.98	187.99 ± 0.98
10	180.32 ± 0.95	191.29 ± 0.98	193.6 ± 0.94	185.09 ± 0.98	188.8 ± 0.98

Table A.16: Value to player two and bounds for each curve shown in Figure 7.6

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges	REFINED -BETS 3 Ranges 50 Extra Bets	REFINED -BETS 3 Ranges 100 Extra Bets
1	152.02 ± 0.94	123.78 ± 0.98	115.52 ± 0.96	116.13 ± 0.98	115.7 ± 0.98
2	141.47 ± 0.93	118.94 ± 0.98	113.18 ± 0.96	113.64 ± 0.98	113.08 ± 0.98
3	137.12 ± 0.94	113.21 ± 0.98	111 ± 0.96	112.13 ± 0.98	111.36 ± 0.98
4	132.34 ± 0.95	113.45 ± 0.97	109.79 ± 0.96	110.99 ± 0.98	111.31 ± 0.98
5	130.24 ± 0.97	112.34 ± 0.97	107.65 ± 0.95	110.97 ± 0.98	110.69 ± 0.98
6	128.27 ± 0.98	113.46 ± 0.97	106.97 ± 0.98	111.61 ± 0.98	111.59 ± 0.98
7	127.63 ± 0.97	112.39 ± 0.97	108.53 ± 0.98	111.9 ± 0.98	111.95 ± 0.98
8	127.25 ± 0.97	112.42 ± 0.96	108.06 ± 0.98	111.27 ± 0.98	111.83 ± 0.98
9	126.7 ± 0.98	113.35 ± 0.96	108.39 ± 0.98	113.45 ± 0.98	112.31 ± 0.98
10	128.9 ± 0.98	112.71 ± 1.01	108.03 ± 1	114.28 ± 0.98	114.07 ± 0.98

Table A.17: Value to player two and bounds for each curve shown in Figure 7.7

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges	REFINED -BETS 3 Ranges 50 Extra Bets	REFINED -BETS 3 Ranges 100 Extra Bets
1	165.78 ± 0.94	188.28 ± 0.98	189.05 ± 0.93	186.83 ± 0.98	187.95 ± 0.98
2	173.36 ± 0.94	193.67 ± 0.98	192.44 ± 0.94	191.71 ± 0.98	190.89 ± 0.98
3	177.63 ± 0.94	192.96 ± 0.98	194.54 ± 0.94	191.8 ± 0.98	192.74 ± 0.98
4	180.58 ± 0.94	193.3 ± 0.98	195.06 ± 0.94	193.42 ± 0.98	192.3 ± 0.98
5	180.76 ± 0.94	193.06 ± 0.98	195.47 ± 0.94	190.44 ± 0.98	192.92 ± 0.98
6	181.04 ± 0.94	193.17 ± 0.98	195.54 ± 0.94	190.04 ± 0.98	192.52 ± 0.98
7	180.22 ± 0.95	192.98 ± 0.98	195.6 ± 0.94	191.82 ± 0.98	193.32 ± 0.98
8	178.69 ± 0.94	193.39 ± 0.98	194.53 ± 0.94	191.58 ± 0.98	192.92 ± 0.98
9	180.72 ± 0.95	192.63 ± 0.98	194.54 ± 0.94	191.95 ± 0.98	193.12 ± 0.98
10	180.32 ± 0.95	191.29 ± 0.98	193.6 ± 0.94	191.37 ± 0.98	193.02 ± 0.98

Table A.18: Value to player two and bounds for each curve shown in Figure 7.8

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges
1	152.63 ± 0.96	125.88 ± 0.97	119.63 ± 0.98
2	144.37 ± 0.96	121.32 ± 0.98	116.24 ± 0.98
3	136.52 ± 0.96	117.24 ± 0.97	113.19 ± 0.98
4	135.08 ± 0.97	115.45 ± 0.97	113.54 ± 0.98
5	132.66 ± 0.97	117.97 ± 0.97	113.63 ± 0.98
6	131.42 ± 0.98	115.79 ± 0.96	111.6 ± 0.98
7	132.16 ± 0.98	115.9 ± 0.96	113.05 ± 0.98
8	130.57 ± 0.98	116.53 ± 0.96	112.6 ± 0.98
9	130.84 ± 0.98	115.71 ± 0.96	113.05 ± 0.98
10	130.41 ± 0.97	115.72 ± 0.98	112.8 ± 0.98

Table A.19: Value to player two and bounds for each curve shown in Figure 7.9

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 3 Ranges
1	171.81 ± 0.96	191.44 ± 0.98	190.94 ± 0.98
2	177.29 ± 0.97	194.71 ± 0.98	193.72 ± 0.98
3	177.29 ± 0.97	194.7 ± 0.98	191.64 ± 0.98
4	179.63 ± 0.97	193.66 ± 0.98	193.36 ± 0.98
5	180.98 ± 0.96	193.32 ± 0.98	195.25 ± 0.98
6	180.91 ± 0.96	193.18 ± 0.98	195.69 ± 0.98
7	180.97 ± 0.96	192.27 ± 0.98	195.24 ± 0.98
8	180.12 ± 0.96	191.1 ± 0.98	195.35 ± 0.88
9	179.75 ± 0.96	191.84 ± 0.98	195.59 ± 1.08
10	179.76 ± 0.97	192 ± 0.98	195.44 ± 0.78

Table A.20: Value to player two and bounds for each curve shown in Figure 7.10

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 1 Range Bets from IR100s	REDM-BETS 2 Ranges Bets from IR100s
1	152.63 ± 0.96	125.88 ± 0.97	153.98 ± 0.96	127.17 ± 0.98
2	144.37 ± 0.96	121.32 ± 0.98	143.99 ± 0.96	122.27 ± 0.98
3	136.52 ± 0.96	117.24 ± 0.97	139.25 ± 0.96	118.21 ± 0.98
4	135.08 ± 0.97	115.45 ± 0.97	135.3 ± 0.97	118.02 ± 0.97
5	132.66 ± 0.97	117.97 ± 0.97	134.35 ± 0.99	118.79 ± 0.96
6	131.42 ± 0.98	115.79 ± 0.96	132.86 ± 0.99	116.62 ± 0.97
7	132.16 ± 0.98	115.9 ± 0.96	132.34 ± 0.99	116.88 ± 0.97
8	130.57 ± 0.98	116.53 ± 0.96	131.69 ± 0.99	116.17 ± 0.96
9	130.84 ± 0.98	115.71 ± 0.96	131.26 ± 0.98	118.2 ± 0.97
10	130.41 ± 0.97	115.72 ± 0.98	132.6 ± 0.98	116.88 ± 0.97

Table A.21: Value to player two and bounds for each curve shown in Figure 7.11

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REDM-BETS 1 Range Bets from IR100s	REDM-BETS 2 Ranges Bets from IR100s
1	171.81 ± 0.96	191.44 ± 0.98	171.7 ± 0.97	190.68 ± 0.96
2	177.29 ± 0.97	194.71 ± 0.98	177.46 ± 0.97	193.46 ± 0.97
3	177.29 ± 0.97	194.7 ± 0.98	179.39 ± 0.97	196.28 ± 0.97
4	179.63 ± 0.97	193.66 ± 0.98	179.29 ± 0.97	195.69 ± 0.97
5	180.98 ± 0.96	193.32 ± 0.98	179.17 ± 0.96	195.03 ± 0.95
6	180.91 ± 0.96	193.18 ± 0.98	179.3 ± 0.96	193.97 ± 0.96
7	180.97 ± 0.96	192.27 ± 0.98	179.81 ± 0.96	195.01 ± 0.95
8	180.12 ± 0.96	191.1 ± 0.98	178.24 ± 0.95	194.1 ± 0.96
9	179.75 ± 0.96	191.84 ± 0.98	178.14 ± 0.96	194.5 ± 0.96
10	179.76 ± 0.97	192 ± 0.98	176.93 ± 0.96	194.83 ± 0.96

Table A.22: Value to player two and bounds for each curve shown in Figure 7.12

Run #	REDM-BETS 1 Range in IR570s	REDM-BETS 2 Ranges in IR570s	REDM-BETS 1 Range in IR100s	REDM-BETS 2 Ranges in IR100s
1	152.63 ± 0.96	125.88 ± 0.97	152.02 ± 0.94	123.78 ± 0.98
2	144.37 ± 0.96	121.32 ± 0.98	141.47 ± 0.93	118.94 ± 0.98
3	136.52 ± 0.96	117.24 ± 0.97	137.12 ± 0.94	113.21 ± 0.98
4	135.08 ± 0.97	115.45 ± 0.97	132.34 ± 0.95	113.45 ± 0.97
5	132.66 ± 0.97	117.97 ± 0.97	130.24 ± 0.97	112.34 ± 0.97
6	131.42 ± 0.98	115.79 ± 0.96	128.27 ± 0.98	113.46 ± 0.97
7	132.16 ± 0.98	115.9 ± 0.96	127.63 ± 0.97	112.39 ± 0.97
8	130.57 ± 0.98	116.53 ± 0.96	127.25 ± 0.97	112.42 ± 0.96
9	130.84 ± 0.98	115.71 ± 0.96	126.7 ± 0.98	113.35 ± 0.96
10	130.41 ± 0.97	115.72 ± 0.98	128.9 ± 0.98	112.71 ± 1.01

Table A.23: Value to player two and bounds for each curve shown in Figure 7.13

Run #	REDM-BETS 1 Range in IR570s	REDM-BETS 2 Ranges in IR570s	REDM-BETS 1 Range in IR100s	REDM-BETS 2 Ranges in IR100s
1	171.81 ± 0.96	191.44 ± 0.98	165.78 ± 0.94	188.28 ± 0.98
2	177.29 ± 0.97	194.71 ± 0.98	173.36 ± 0.94	193.67 ± 0.98
3	177.29 ± 0.97	194.7 ± 0.98	177.63 ± 0.94	192.96 ± 0.98
4	179.63 ± 0.97	193.66 ± 0.98	180.58 ± 0.94	193.3 ± 0.98
5	180.98 ± 0.96	193.32 ± 0.98	180.76 ± 0.94	193.06 ± 0.98
6	180.91 ± 0.96	193.18 ± 0.98	181.04 ± 0.94	193.17 ± 0.98
7	180.97 ± 0.96	192.27 ± 0.98	180.22 ± 0.95	192.98 ± 0.98
8	180.12 ± 0.96	191.1 ± 0.98	178.69 ± 0.94	193.39 ± 0.98
9	179.75 ± 0.96	191.84 ± 0.98	180.72 ± 0.95	192.63 ± 0.98
10	179.76 ± 0.97	192 ± 0.98	180.32 ± 0.95	191.29 ± 0.98

Table A.24: Value to player two and bounds for each curve shown in Figure 7.14

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges
1	180.26 ± 0.95	155.04 ± 0.98
2	175.3 ± 0.96	152.57 ± 0.98
3	170.87 ± 0.96	149.02 ± 0.98
4	166.57 ± 0.96	148.02 ± 0.98
5	163.19 ± 0.96	147.02 ± 0.98
6	161.72 ± 0.96	149.02 ± 0.98
7	161.24 ± 0.96	148.02 ± 0.98
8	161.77 ± 0.95	149.52 ± 0.98
9	160.82 ± 0.96	148.22 ± 1.19
10	160.63 ± 0.97	147.52 ± 1.38

Table A.25: Value to player two and bounds for each curve shown in Figure 7.15

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges
1	140.37 ± 0.95	159.8 ± 0.98
2	149.74 ± 0.96	161.93 ± 0.98
3	148.49 ± 0.95	161.23 ± 0.98
4	150.47 ± 0.96	159.23 ± 0.98
5	151.73 ± 0.96	158.23 ± 0.98
6	152.43 ± 0.96	157.23 ± 0.98
7	153.98 ± 0.95	159.23 ± 0.98
8	153.85 ± 0.95	158.23 ± 0.98
9	151.01 ± 0.96	159.23 ± 1.18
10	151.55 ± 0.75	157.23 ± 1.38

Table A.26: Value to player two and bounds for each curve shown in Figure 7.16

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REFINED-BETS 3 Ranges 100 Extra Bets
1	180.26 ± 0.95	155.04 ± 0.98	149.06 ± 0.98
2	175.3 ± 0.96	152.57 ± 0.98	147.14 ± 0.98
3	170.87 ± 0.96	149.02 ± 0.98	146.78 ± 0.98
4	166.57 ± 0.96	148.02 ± 0.98	144.29 ± 0.98
5	163.19 ± 0.96	147.02 ± 0.98	144.49 ± 0.98
6	161.72 ± 0.96	149.02 ± 0.98	144.09 ± 0.98
7	161.24 ± 0.96	148.02 ± 0.98	144.59 ± 0.98
8	161.77 ± 0.95	149.52 ± 0.98	144.39 ± 0.98
9	160.82 ± 0.96	148.22 ± 1.19	143.89 ± 0.98
10	160.63 ± 0.97	147.52 ± 1.38	144.89 ± 0.98

Table A.27: Value to player two and bounds for each curve shown in Figure 7.17

Run #	REDM-BETS 1 Range	REDM-BETS 2 Ranges	REFINED-BETS 3 Ranges 200 Extra Bets
1	140.37 ± 0.95	159.8 ± 0.98	156.52 ± 0.98
2	149.74 ± 0.96	161.93 ± 0.98	158.78 ± 0.98
3	148.49 ± 0.95	161.23 ± 0.98	160.45 ± 0.98
4	150.47 ± 0.96	159.23 ± 0.98	159.5 ± 1
5	151.73 ± 0.96	158.23 ± 0.98	160 ± 1.2
6	152.43 ± 0.96	157.23 ± 0.98	161 ± 1
7	153.98 ± 0.95	159.23 ± 0.98	159.5 ± 1.1
8	153.85 ± 0.95	158.23 ± 0.98	158.5 ± 1.2
9	151.01 ± 0.96	159.23 ± 1.18	159.3 ± 1
10	151.55 ± 0.75	157.23 ± 1.38	159.8 ± 1.1

Table A.28: Value to player two and bounds for each curve shown in Figure 7.18

Bibliography

[1] The Computer Poker Competition. http://www.computerpokercompetition.org/.

[2] The Second Man-Machine Poker Competition. http://manmachinepoker.com.

[3] N. Abou Risk. Using Counterfactual Regret Minimization to Create a Competitive Multiplayer Poker Agent. Master's thesis, University of Alberta, 2009.

[4] A. Antos, R. Munos, and C. Szepesvari. Fitted Q-iteration in Continuous Action-Space MDPs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 9–16, 2007.

[5] N. Bard, M. Johanson, N. Burch, and M. Bowling. Online Implicit Agent Modelling. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 255–262, 2013.

[6] D. Billings. Algorithms and Assessment in Computer Poker. Master's thesis, University of Alberta, 2006.

[7] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating Game-Theoretic Optimal Strategies for Full-scale Poker. In *Proceedings of the Eighteenth International Joint Conference on Artifical Intelligence (IJCAI)*, 2003.

[8] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The Challenge of Poker. *Artificial Intelligence Journal*, 2002.

[9] B. Chen and J. Ankenman. *The Mathematics of Poker*. ConJelCo Publishing Company, 2007.

[10] M. Craig, editor. *The Full Tlit Poker Strategy Guide*. Grand Central Publishing, 2007.

[11] B. David. An Analog of the Minimax Theorem for Vector Payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956.

[12] T. Dean, K. Kim, and R. Givan. Solving Stochastic Planning Problems with Large State and Action Spaces. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 102–110, 1998.

[13] M. Flynn, S. Mehta, and E. Miller. *Professional No-limit Hold'em*. Two Plus Two Publishing, 2007.

[14] S. Ganzfried and T. Sandholm. Computing an Approximate Jam/Fold Equilibrium for 3-Agent No-Limit Texas Hold'em Tournaments. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

[15] S. Ganzfried and T. Sandholm. Computing Equilibria in Multiplayer Stochastic Games of Imperfect Information. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

[16] S. Ganzfried and T. Sandholm. Computing Equilibria by Incorporating Qualitative Models. In *Proceedings of the National Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[17] S. Ganzfried, T. Sandholm, and K. Waugh. Strategy Purification and Thresholding: Effective non-Equilibrium Approaches for Playing Large Games. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 871–878, 2012.

[18] R. Gibson, M. Lanctot, N. Burch, D. Szafron, and M. Bowling. Generalized Sampling and Variance in Counterfactual Regret Minimization. In *Proceedings of the Twenty-Sixth International Conference on Artificial Intelligence (AAAI)*, pages 1355–1361, 2012.

[19] R. Gibson and D. Szafron. On Strategy Stitching in Large Extensive Form Multiplayer Games. In *Advances in Neural Information Processing Systems (NIPS)*, pages 100–108, 2011.

[20] A. Gilpin, J. Pena, and T. Sandholm. First-Order Algorithm with O(ln(1/epsilon)) Convergence for epsilon-Equilibrium in Two-Person Zero-Sum Games. In *Proceedings of the International Conference on Artificial Intelligence (AAAI)*, 2008.

[21] A. Gilpin and T. Sandholm. A Competitive Texas Hold'em Poker Player via Automated Abstraction and Real-time Equilibrium Computation. In *Proceedings of the International Conference on Artificial Intelligence (AAAI)*, 2006.

[22] A. Gilpin and T. Sandholm. Better Automated Abstraction Techniques for Imperfect Information Games, with Application to Texas Hold'em Poker. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2007.

[23] A. Gilpin and T. Sandholm. Gradient-based Algorithms for Finding Nash Equilibria in Extensive Form Games. In *Proceedings of the Third International Workshop on Internet and Network Economics (WINE)*, 2007.

[24] A. Gilpin, T. Sandholm, and T. Soerensen. Potential-aware Automated Abstraction of Sequential Games, and Holistic Equilibrium Analysis of Texas Hold'em Poker. In *Proceedings of the International Conference on Artificial Intelligence (AAAI)*, 2007.

[25] A. Gilpin, T. Sandholm, and T. B. Sorensen. A Heads-up No-limit Texas Hold'em Poker Player: Discretized Betting Models and Automatically Generated Equilibrium-finding Programs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

[26] H. Gintis. *Game Theory Evolving*. Princeton University Press, 2000.

[27] D. Harrington and B. Robertie. *Harrington on Hold'em Volume I*. Two Plus Two Publishing, 2005.

[28] D. Harrington and B. Robertie. *Harrington on Hold'em Volume II*. Two Plus Two Publishing, 2005.

[29] D. Harrington and B. Robertie. *Harrington on Cash Games Volume I*. Two Plus Two Publishing, 2008.

[30] D. Harrington and B. Robertie. *Harrington on Cash Games Volume II*. Two Plus Two Publishing, 2008.

[31] H. V. Hasselt and M. Wiering. Reinforcement Learning in Continuous Action Spaces. In *IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, 2007.

[32] J. Hawkin, N. Bard, J. Rubin, and M. Zinkevich. The Annual Computer Poker Competition. *AI Magazine*, 34(2):112–114, 2013.

[33] J. Hawkin, R. Holte, and D. Szafron. Using Sliding Windows to Generate Action Abstractions in Extensive-Form Games. In *Proceedings of the Twenty-sixth International Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1924–1930, 2011.

[34] J. Hawkin, R. Holte, and D. Szafron. Automated Action Abstraction of Imperfect Information Extensive-form Games. In *Proceedings of the Twenty-seventh International Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, pages 681–687, 2012.

[35] B. Hoehn. The Effectiveness of Opponent Modelling in a Small Imperfect Information Game. Master's thesis, University of Alberta, 2006.

[36] E. Jackson. Slumbot: An Implementation Of Counterfactual Regret Minimization On Commodity Hardware. In *Computer Poker Symposium at the Twenty-Sixth Conference on Artificial Intelligence (AAAI)*, 2012.

[37] M. Johanson. Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player. Master's thesis, University of Alberta, 2007.

[38] M. Johanson. Measuring the Size of Large No-limit Poker Games. Technical report, University of Alberta, Department of Computing Science, 02 2013.

[39] M. Johanson, N. Bard, N. Burch, and M. Bowling. Finding Optimal Abstract Strategies in Extensive Form Games. In *Proceedings of the Twenty-Sixth International Conference on Artificial Intelligence (AAAI)*, pages 1371–1379, 2012.

[40] M. Johanson, N. Bard, M. Lanctot, R. Gibson, and M. Bowling. Efficient Nash Equilibrium Approximation through Monte Carlo Counterfactual Regret Minimization. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 837–844, 2012.

[41] M. Johanson, M. Bowling, K. Waugh, and M. Zinkevich. Accelerating Best Response Calculation in Large Extensive Games. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 258–265, 2011.

[42] M. Johanson, N. Burch, R. Valenzano, and M. Bowling. Evaluating State-Space Abstractions in Extensive-Form Games. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 271–278, 2013.

[43] D. Koller, N. Megiddo, and B. V. Stengel. Fast Algorithms for Finding Randomized Strategies in Game Trees. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, 1994.

[44] H. Kuhn. Extensive Games and the Problem of Information. *Annals of Mathematics Studies, Volume 28*, 1953.

[45] H. W. Kuhn. Simplified Two-Person Poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

[46] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling. Monte Carlo Sampling for Regret Minimization in Extensive Games. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1078–1086, 2009.

[47] A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement Learning in Continuous Action Spaces through Sequential Monte Carlo Methods. In *Advances in Neural Information Processing Systems (NIPS)*, pages 833–840, 2007.

[48] C. Madeira, V. Corruble, and G. Ramalho. Designing a Reinforcement Learning-based Adaptive AI for Large-scale Strategy Games. In *Proceedings of the Second International Conference On Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 121–123, 2006.

[49] S. Mehrotra. On the Implementation of a Primal-dual Interior Point Method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[50] P. B. Miltersen and T. B. Sorensen. A Near-Optimal Strategy for a Heads-up No-limit Texas Holdem Poker Tournament. In *AAMAS '07: Proceedings of The 6th International Conference on Autonomous Agents and Multiagent Systems*, 2007.

[51] C. Moshman. *Heads-up No-limit Hold'em*. Two Plus Two Publishing, 2008.

[52] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7(4):308–313, 1965.

[53] Y. Nesterov. Excessive Gap Technique in Non-smooth Convex Minimization. *SIAM J. on Optimization Volume 16*, 2005.

[54] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[55] J. Rubin and I. Watson. Similarity-Based Retrieval and Solution Re-use Policies in the Game of Texas Hold'em. In *Proceedings of the Eighteenth International Conference on Case-Based Reasoning (ICCBR)*, pages 465–479, 2010.

[56] J. Rubin and I. Watson. Computer poker: a review. *Artificial Intelligence*, 175(5):958–987, 2011.

[57] J. Rubin and I. Watson. Successful Performance via Decision Generalisation in No Limit Texas Hold'em. In *Proceedings of the Nineteenth International Conference on Case-Based Reasoning (ICCBR)*, pages 467–481, 2011.

[58] D. Schnizlein. State Translation in No-limit Poker. Master's thesis, University of Alberta, 2009.

[59] D. Schnizlein, M. Bowling, and D. Szafron. Probabilistic State Translation in Extensive Games with Large Action Sets. In *Proceedings of the Twenty-first International Joint Conference on Artifical Intelligence (IJCAI)*, 2009.

[60] I. Schweizer, K. Panitzek, S. Park, and J. Fürnkranz. An Exploitative Monte-Carlo Poker Agent. In *Proceedings of the Thirty-second Annual German Conference on AI*, 2009.

[61] D. Sklansky. *The Theory of Poker*. Two Plus Two Publishing, 1987.

[62] D. Sklansky and E. Miller. *No-limit Hold'em: Theory and Practice*. Two Plus Two Publishing, 2006.

[63] N. Sweeney and D. Sinclair. Learning to Play Multi-Player Limit Hold'em Poker by Stochastic Gradient Ascent. In *Proceedings of the Sixth European Workshop on Multi-agent Systems (EUMAS)*, 2009.

[64] S. Tijs. Stochastic Games with One Big Action Space in Each State. Open Access Publications from Tilburg University, Tilburg University, 1980.

[65] K. Waugh. Abstraction in Large Extensive Games. Master's thesis, University of Alberta, 2009.

[66] K. Waugh, D. Schnizlein, M. Bowling, and D. Szafron. Abstraction Pathologies in Extensive Games. In *AAMAS '09: Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[67] K. Waugh, M. Zinkevich, M. Johanson, M. Kan, D. Schnizlein, and M. Bowling. A Practical Use of Imperfect Recall. In *Proceedings of the Eighth Symposium on Abstraction, Reformulation and Approximation (SARA)*, 2009.

[68] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret Minimization in Games with Incomplete Information. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.