Time and Space: Why Imperfect Information Games are Hard

by

Neil Burch

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

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Decision-making problems with two agents can be modeled as two player games, and a Nash equilibrium is the basic solution concept describing good play in adversarial games. Computing this equilibrium solution for imperfect information games, where players have private, hidden information, is harder than solving perfect information games. Both domains may technically be classified as easy, with algorithms that require polynomial time and space, but the difference in degree means that for extremely large games, both computation time and storage requirements may exceed available resources.

In this thesis, I present four main contributions towards fast and memory efficient techniques for solving extensive-form games, which describe a class of imperfect information games with sequential decision making. First, the thesis introduces an analysis of counterfactual regret minimisation (CFR), an algorithm for solving extensive-form games, and presents tighter regret bounds that describe the rate of progress. CFR is a popular, state-of-the-art algorithm, and the improved bounds give some indication as to why the algorithm has good practical performance.

Second, I describe and analyse a game-solving algorithm, CFR$^+$, which has faster empirical performance than CFR, and compare it to a number of theoretically faster algorithms. We wrote and released an efficient, distributed implementation of CFR$^+$ to solve heads-up limit Texas hold'em, making it the first competitively played imperfect information game to be solved.

Third, the thesis presents a series of theoretical tools for using decomposition, and creating algorithms which operate on small portions of a game at a time. I describe an algorithm, CFR-D, which can solve games without ever storing a complete strategy, and an algorithm for re-solving a portion of a game to complete an incomplete equilibrium within the full game. Both algorithms have a formal guarantee of correctness.

Finally, I describe continual resolving, an algorithm which is an imperfect information analogue to heuristic search in perfect information games. Continual re-solving uses decomposition to play games by doing depth-limited solving with a heuristic evaluation, with a theoretical bound on solution quality. We implemented continual re-solving in the game of no-limit heads-up no-limit Texas hold'em, and beat a group of 33 professional players in a human trial.

Preface

The DeepStack research project discussed in Chapter 6 received ethics approval from the University of Alberta Research Ethics Board, Project Name "Evaluating the skill of state-of-the-art computer poker agents", No. 67663, October 24 2016.

Portions of Chapter 3 were based on work published as R. Gibson, N. Burch, M. Lanctot, and D. Szafron, "Efficient Monte Carlo Counterfactual Regret Minimization in Games with Many Player Actions," *Advances in Neural Information Processing Systems 25.*

Portions of Chapter 4 were published as M. Bowling, N. Burch, M. Johanson, and O. Tammelin, "Heads-Up Limit Hold'em Poker is Solved," *Science*, vol. 347, issue 6218, 145-149. I was responsible for significant portions of the implementation, along with the M. Johanson and O. Tammelin. M. Johanson and I performed preliminary experiments. M. Bowling led the research project, and was responsible for manuscript composition. M. Johanson and I assisted with composition and editing.

Portions of Chapter 4 were published as O. Tammelin, N. Burch, M. Johanson, and M. Bowling, "Solving Heads-up Limit Texas Hold'em," *Proceedings of the Twenty-Fourth International Joint Conference on Computers and Games.* I was responsible for experiments, along with M. Johanson. I provided the theoretical results, and was responsible for manuscript composition.

Portions of Chapter 5 were published as N. Burch, M. Johanson, and M. Bowling, "Solving Imperfect Information Games Using Decomposition," *Proceedings of the Twenty-Eight AAAI Conference on Artificial Intelligence.*

Portions of Chapter 6 were published as M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and

M. Bowling. “DeepStack: Expert-level Artificial Intelligence in Heads-up No-limit Poker,” *Science*, vol. 356, issue 6337, 508-513. I was responsible for the theoretical bounds, along with T. Davis. I provided an initial experimental framework, and generated one data set used to train the evaluation function. M. Bowling led the research project, and was responsible for manuscript composition. I assisted with composition and editing, along with all other project members.

I laughed. I cried. I fell down. It changed my life.

– Stephen Brust, *Cowboy Feng's Space Bar and Grille*

Acknowledgements

I would like to thank my supervisor, Michael Bowling, for his advice on writing, presenting, teaching, research, collaboration, and setting reasonable expectations. I am grateful for his support throughout my studies, and his unwavering attention to what is fair and reasonable. I am also grateful for his unflagging efforts in bringing together everything needed for large, exciting research projects.

I would like to thank all the members of the Computer Poker Research group, past and present. You were my collaborators on things that worked, and shared the pain when the next great idea did not.

I am grateful to Jonathan Schaeffer for the opportunity to slowly realise how much I wanted to do games and artificial intelligence research.

From the large, like financial support through scholarships and a great learning environment with excellent faculty members, to the small, like finding hardware parts or letting me into the office after locking myself out, thank you to the University of Alberta, and all of the support staff and faculty in the Computing Science department.

Finally, thank you to my parents George and Angelina Burch, and my wife Joanne Wojtysiak for their support, love, and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

From its inception, artificial intelligence research has been used to build computer agents that play games. Babbage had plans for a machine that would play tic-tac-toe [2]. Both Borel and von Neumann did theoretical work on games, using poker as an example and motivation [6, 90, 91]. The reason is that teaching a computer to play games is not frivolous, even if it is fun. A game is a decision making problem, with clear rules and concrete way to evaluate performance.

The challenges that appear in single-player puzzles like Sokoban also appear in problems like pathfinding and scheduling. The performance of heuristic search algorithms for games like chess and checkers depends on the quality of the heuristic evaluation [93], which is an example of function approximation for a very complicated function. IBM's Jeopardy-playing program Watson [25] demonstrated advances in understanding natural language, and inference on databases of knowledge about the world. DeepMind's work on Atari [64] modeled the game as a reinforcement learning problem, an approach which has also been applied to motion control for robots.

These examples of success share a common factor: agents do not have private, hidden information that is intrinsically part of the decision making problem. You might have knowledge about good checkers strategy that I do not have, and neither of us know what question lies behind a square in Jeopardy, but there is nothing built into these situations where you know something I do not as part of the rules of play. If both players are good enough, there is no reason to deviate from honest play where both players pick the best action.

There is another class of problems, described by von Neumann: "Real life is not like that. Real life consists of bluffing, of little tactics of deception, of asking yourself what is the other man going to think I mean to do. And that is what games are about in my theory." [12] As suggested by von Neumann's quote, poker is one of these games where players have imperfect information.

In an imperfect information game, players have knowledge about the current state of the game that is not available to their opponent. A player has to reason under that uncertainty, but they can also act so as to take advantage of their opponent's uncertainty. Those two factors, added to the circular reasoning that defines optimal play, make it harder to choose a strategy for problems with imperfect information than it is to choose a strategy for problems with perfect information. Solving a problem is no longer a matter of picking the best action at each situation, but picking a randomised, non-deterministic strategy which balances exploiting our knowledge of what we think the opponent will do, and deceiving our opponent by hiding our own information.

Early techniques [88] used a representation that grew in size exponentially in the number of decision points, and could only be applied to very small problems. Working out perfect play in a game with just 30 binary decisions for each player would require a billion by billion matrix to represent the game, and result in a randomised strategy which samples from a billion possibilities. Storing the matrix requires too much space, and even if we could store it, a computation that looked at each entry would take too much time.

Using too much time, too much space, or both has continued to characterise algorithms for solving imperfect information games. Koller *et al.* introduced an algorithm whose space and running time is polynomial in the number of decisions [50, 51], letting us solve non-trivial games. Despite this enormous improvement, the quadratic space requirements are too much for games like poker, where the smallest commonly-played variant has 10^{14} decisions.

The next improvements were an algorithm by Zinkevich *et al.* called counterfactual regret minimisation, that used the online learning technique of regret minimisation [94], and an algorithm by Gilpin *et al.* based on accelerated gradient descent methods [31]. Both algorithms produce a strategy which is an approximation of optimal play, but can be implemented in a way that uses linear space. The reduced space requirements, however, expose a worse-than-cubic computation time.

The work described in this thesis is a series of practical and theoretical improvements, pushing at the boundaries of space and time requirements.

- In Chapter 3, I give improved theoretical bounds on performance of the CFR algorithm of Zinkevich *et al.*, helping to explain the good empirical performance of CFR in large games like poker.
- In Chapter 4, I provide performance bounds for CFR$^+$, a new, faster variant of CFR.
- In Chapter 4, I prove that the component algorithm regret-matching$^+$ used by CFR$^+$ has an online learning property not shared by commonly used online learning algorithms.
- In Chapter 5, I introduce theoretical tools for saving space by decomposing imperfect information games into smaller problems.
- In Chapter 6, I give performance bounds for continual re-solving, a new algorithm which uses both CFR and decomposition to get a theoretically sound analogue of perfect information heuristic search.

The CFR$^+$ and continual re-solving algorithms both led to landmark results. I was a significant part of a group effort using CFR$^+$ to solve the game of heads-up limit Texas hold'em poker, making it the first commonly played imperfect information game to be solved. I was also a significant part of a larger collaborative project using continual re-solving to play the more complicated no-limit variant of Texas hold'em poker, and test it in a human study. We had strong positive results, and our program DeepStack was the first to beat professional human players at the no-limit game.

Chapter 2

Background

My research requires terminology from a number of areas. I use poker as an experimental testbed, and game theory is the appropriate theoretical framework for the game of poker. One of my main tools for dealing with games rests on the online learning framework of regret minimisation, using one particular type of expected value called counterfactual value. The background terminology in this chapter will be used throughout the chapters that follow.

2.1 Poker

Most of my work is experimentally verified using Texas hold'em, the most popularly played variant of poker. The different variants of poker vary widely, but they are all adversarial card games where players bet on having the better set of cards, called their hand. The player's hand is a subset of their own private cards, often called hole cards, and shared public cards, often called board cards. The player's hand can also refer to just their private cards.

Each player has a certain number of chips called their stack, and each game starts with a number of forced initial bets of fixed size, called the ante or the blinds. There are then one or more betting rounds where some number of private and public cards are dealt, and then players take turns making betting actions. The anted chips, and any subsequent bets, go into a pool of chips called the pot.

A round of betting proceeds by players choosing to either raise the current bet by placing more chips in the pot, call the bet by matching the largest contribution to the pot, or fold their hand. If no bet has yet been made in the round, a player is said to check rather than call. The first bet in a round is said to be a bet, while subsequent betting actions are called raises. A round of betting is over when all players have checked, or called the last bet or raise.

A player that folds forfeits their claim to the pot, and stops participating in the game. If there is only one player still participating, the game is over early and that player wins all of the chips in the pot. Otherwise, the game ends in a showdown after the last betting round is finished. In the showdown, players that are still participating show their private cards[1] and the player with the best hand claims all chips in the pot. In the case where there is a tie, the chips in the pot are split between the tied players.

In poker as it is usually played by humans, the player's winnings from the previous game[2] carry forward into the next game by affecting the player's stacks. This means the games are not independent, and are not a single repeated game. My work focuses on a variant called Doyle's game [34] where the player's stacks are reset at the beginning of each game. In this case, each game of poker is an independent instance of the same repeated game.

2.1.1 Heads-up Texas Hold'em

Texas hold'em is a popular variant of poker that has four rounds of betting. Players get two private cards in the first round, called the pre-flop. There are three public cards added to the board in the second round, called the flop. One more public card is added to the board in each of the third and fourth rounds, called the turn and river, respectively.

In Texas hold'em, the initial forced betting actions are known as the small blind and the big blind, where the big blind is twice as large as the small blind. Heads-up Texas hold'em, the two player variant, is played with what is known as reverse blinds, where the small blind player acts first on the first round and the big blind player acts first in all subsequent rounds.

[1] A participating player might also choose to not show their cards, and forfeit their claim to the pot, as if they had folded.

[2] Each game is usually called a hand, although I avoid this usage.

When played by humans, at the end of each game, players swap between the small and big blind. This is because there is a positional advantage, and swapping ensures that a player does not always have the disadvantageous position. From the point of view of game theory and this document, the term player will refer to the position within the game and not an agent that swaps between positions. That is, the first player will act first in every game.

Texas hold'em has two betting variants: limit and no-limit. In limit games, there is a fixed size for bets and raises. Players can bet or raise the same number of chips as the big blind in the pre-flop and flop rounds, and twice the big blind in the last two rounds. In no-limit games, players can bet or raise any number of chips in a range going from the last bet or raise in that round (including the blinds) up to their entire stack. Betting all remaining chips in the stack is known as going all-in. I will refer to heads-up limit Texas hold'em as HULHE, and heads-up no-limit Texas hold'em as HUNL. HULHE is the smallest variant of poker that is commonly played by humans, with around $3.19 * 10^{14}$ information sets [9].

Because the big blind determines the bet sizes in HULHE and the minimum bet size in HUNL, it is often more meaningful to use big blinds as a unit rather than directly referring to chips. Winning 2000 chips after 1000 games says something very different about my skill when the game had a 2 chip big blind, than it does if the game had a 100 chip big blind: the former is an astoundingly large win-rate of 1 big blind per game, while the latter is a modest (but satisfying) 0.02 big blinds per game. For smaller numbers, I will use mbb to denote a thousandth of a big blind. We could then say my 0.02 big blind per game win rate is 20 mbb/g.

2.2 Imperfect Information

Games are decision making problems with fully described rules, specifying the available actions and the values for all possible outcomes. Because games are well-specified problems, we can partition them into a number of theoretically different classes. One of the biggest distinguishing features is the number of players: single-player games, two-player games, and games with more than two players all have different properties. Another important distinction can be made between games where the players have identical information about the current state, and those where players have different information about the state.

Informally, the distinction between perfect information and imperfect information is often made in terms of whether all players exactly know the current game state. For example, chess and checkers are perfect information games, where the board and moves are all public information. In imperfect information games, players may not know something about the current state. For example, in rock-paper-scissors, we can say one player acts first and the second player has incomplete information about the state and must choose an action without knowing the first player's action. In poker, all actions are made without knowledge of the opponent's privately held cards.

The informal "full knowledge" distinction is less useful when considering the presence or lack of non-deterministic events, also called chance or random events. The unknown outcome of dice rolls in backgammon or the not yet revealed cards in solitaire games do not make those games imperfect information games. Similarly, the inclusion of unknown face-down board cards in poker games does not make them imperfect information games. It is the player's private cards that create the necessary information asymmetry. To be an imperfect information game, there must be at least one situation where a player makes an action without knowing the same information as the player making some earlier action. However, as long as we consider chance events to be undetermined until some player learns the result, the informal "doesn't know the exact state" version of imperfect information is a satisfactory definition.

Reasoning about imperfect information games is different than reasoning about perfect information games, because it requires circular reasoning. When we act, we must think about the possible distribution over the unknown information that was used to make earlier actions. In turn, the earlier actions must be made with an understanding of what information they reveal.

For example, the decision to bluff by betting with weak cards is a natural

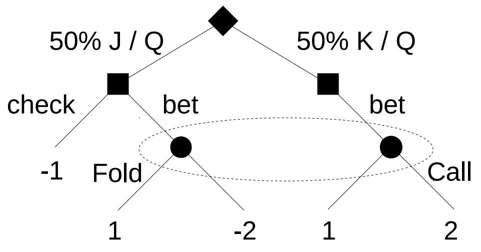


Figure 2.1: Simplified poker situation which requires bluffing

consequence of the rules of poker. Consider the simplified poker situation in Figure 2.1. The game starts with a chance state represented by a diamond, where we have equal chance of being dealt J or K and the opponent is always dealt Q. Our decision, represented by the squares, is whether we should bet or check. We always bet with K and are choosing whether to check or bet with J. At the circles, our opponent is choosing whether to fold or call to a bet. Our opponent does not know our cards, an information set represented by the dashed oval containing the two circles. Betting with J is a bluff, as we lose by making the action unless the opponent responds as if we had K.

If we never bet with J, then our opponent knows that if we bet we must have K, and will maximise their value by folding. Our expected value in this case is $0.5 * (-1 + 1) = 0$. If we always bet with J, then our opponent knows that if we bet that it's equally likely we have J or K, and will maximise their value by calling. Our expected value in this case is $0.5 * (-2 + 2) = 0$. However, if we bluff by betting J with probability 1/3, our opponent is ambivalent about folding or calling, and our expected value is 1/3. Done carefully, misrepresenting our cards increases our expected value.

The bluffing behaviour in poker also highlights another difference between perfect information games and imperfect information games. Optimal play in perfect information games can be described by picking one single best action at every state, whereas optimal play in imperfect information game usually requires the players to act stochastically.

2.3 Extensive Form Games

An extensive form game is a tree-based formalism used to describe a large class of imperfect information games[3]. As an extension of the classical idea of a game tree, extensive form games are also capable of describing perfect information games: perfect information games are a theoretically trivial (but not practically trivial!) subset of imperfect information games.

Formally speaking, an extensive form game $\mathcal{G}$ is a tuple $\langle H, Z, P, p, u, \mathcal{I}, \sigma_c \rangle$. H is a set of states, including $\varnothing$, the initial state of the game. A state can also be called a history, because a game state is exactly a history of the sequence of actions taken from the initial state. I will use $\cdot$ to indicate concatenation, so $h \cdot a$ is the sequence of actions in h, followed by action a. $Z \subset H$ is the set of terminal (leaf) states, and $u_p : Z \mapsto \mathbb{R}$ gives the payoff to player p if the game ends at state z.

P is a set of all players acting in the game, and $p : H \setminus Z \mapsto P$ is a function which describes which player is currently acting at a non-terminal state h. The actions available at a non-terminal state h are defined implicitly by the set of histories H, so that $A(h) = \{a | h \cdot a = j, h \in H, j \in H\}$. For a history h where a stochastic event is about to happen, like a die roll or card deal, $p(h)$ is a special "chance player" c. The value $\sigma_c(h, a)$ gives the probability of chance event a occurring if the game is in state h.[4]

$\mathcal{I}$ describes what information is hidden from the players in a game, defined by a partition of all non-terminal, non-chance states. $\mathcal{I}$ must satisfy the constraints that $\forall I \in \mathcal{I}$ and $\forall h, j \in I$, we have $p(h) = p(j)$ and $A(h) = A(j)$. A set $I \in \mathcal{I}$ is called an information set, and for every state $h \in I$ the player $p(h)$ will only know that the current state of the game is one of the states in I, but not exactly which one.

There are a number of notations of convenience. Because the player and

[3]Whether or not a decision making problem can be described as an extensive form game depends on details like whether we allow trees to have an uncountably infinite number of branches, or infinitely long branches.

[4]By convention, a game never ends in a chance event that only leads to terminal states, because we can construct a strategically identical game by using a payoff which is an expectation over the terminal chance event.

available actions are the same for all states in any information set I, we can define $p(I) = p(h)$ and $A(I) = A(h)$ for an arbitrary $h \in I$. We can refer to the set of information sets for a player p as $\mathcal{I}_p = \{I \in \mathcal{I} | p(I) = p\}$. Finally, it is convenient to speak of $I \cdot a = \{h \cdot a | h{\in}I\}$, the set of states that a player can be in after taking action a at information set I.

We will say that $h \sqsubseteq j$, or h is an ancestor of j, if h is a prefix of j. $h \sqsubset j$ and h is a strict ancestor of j if $h \sqsubseteq j$ and $h \neq j$. h is the parent of j if $j = h \cdot a$ for some action a, and j is a child of h if h is the parent of j. We will let $Z(S)$ be the set of terminal histories that can be reached from a set of states S. Conversely, given a history h and set of states S, $h[S]$ is the longest prefix of h that is in set S, or $\varnothing$ if no such prefix exists.

2.3.1 Zero Sum

A game is said to be constant-sum if $\sum_{p \in P} u_p(z) = k \in \mathbb{R}$ for all $z \in Z$, and zero-sum if the game is constant-sum with $k = 0$. These concepts are functionally equivalent, because a game is strategically identical if we shift or scale all payoffs by a constant value. The work in this thesis assumes a zero-sum game.

2.3.2 Perfect Recall

Informally, a player is said to have perfect recall if they do not forget any information they previously knew. Formally, a player p has perfect recall if for all information sets $I \in \mathcal{I}_p$ and any states $h, j \in I$, h and j have passed through the same sequence of player p information sets, and made the same action at those information sets. That is, for any state s there is a sequence of states $s_0 = \varnothing, ..., s_m$ such that $s_{i+1} = s_i \cdot a_i$ and $s_m \cdot a_m = s$, which induces a sequence of tuples $\langle I_i, a_i \rangle$ such that $s_i \in I_i$, from which we can get a subsequence of tuples $\langle I_i, a_i \rangle$ where $p(I_i) = p(s)$. If this subsequence is identical for all $h, j \in I$ and all $I \in \mathcal{I}_p$, player p has perfect recall.

If a player does not have perfect recall, they are said to have imperfect recall. If all players have perfect recall, the game is said to be a perfect-recall game. All of the work in this thesis assumes a perfect-recall game.

2.4 Player Strategies

A player's strategy, also known as their policy, determines how they choose actions at states where they are acting. The term strategy profile is used to refer to a tuple consisting of a strategy for each player. I will use σ_p to refer to a strategy for player p, and σ to refer to a strategy profile. The set of all player p strategies will be denoted by Σ_p, and the set of strategy profiles will be denoted as Σ.

Given a strategy profile σ and some player p strategy σ'_p, I will use the tuple $< \sigma_{-p}, \sigma'_p >$ to refer to the strategy profile where player p's strategy is σ'_p, and their opponent plays according to their strategy in σ.

2.4.1 Strategy Probabilities

It is useful to have terminology for frequently used probabilities. $\sigma_p(I, a)$ gives the probability of player p making action a given they have reached information set $I \in \mathcal{I}_p$, and $\sigma(I, a) = \sigma_{p(I)}(I, a)$. I will use the vector $\sigma(I)$ to speak of the probability distribution $\sigma(I, a)$ over $A(I)$. $\pi^\sigma(h)$ gives the probability of reaching state h if all players follow profile σ.

There are many more common conditional probabilities. $\pi^\sigma_p(h)$ gives the probability of reaching state h if chance and p's opponents make the actions to reach h, and player p acts according to σ. In perfect recall games, we can extend this to speak of $\pi^\sigma_p(I)$ as all states in I will have the same player p probability.

We can also extend the π notation by flipping which players are following σ and use $\pi^\sigma_{-p}(h)$ to refer to the probability of reaching state h if player p makes the actions to reach h, and chance and p's opponents act according to σ. All of these π probabilities can also be extended to consider subsequences of actions. We can speak of $\pi^\sigma_p(z|h)$ as the probability of player p making the actions needed to move from state h to state z.

In perfect recall games, all of the π probabilities can be expressed as products of $\sigma(I^k, a)$ terms. This gives us equivalences like $\pi^\sigma(h) = \pi^\sigma_p(h)\pi^\sigma_{-p}(h)$ and $\pi^\sigma(z) = \pi^\sigma(h)\pi^\sigma(z|h)$ for any $h \sqsubset z$. We will define $\pi^\sigma_p(I, a) = \pi^\sigma_p(I)\sigma(I, a)$.

$$\begin{tikzcd}
\varnothing \arrow[r, "3\clubsuit\mathrm{A}\clubsuit/7\spadesuit\mathrm{Q}\heartsuit"'] & h_1 \arrow[r, "\mathrm{c}"'] & h_2 \arrow[r, "\mathrm{B}"'] & h_3 \arrow[r, "\mathrm{c}"'] & h_4 \arrow[r, "2\clubsuit 4\diamondsuit \mathrm{Q}\clubsuit"'] & h_5 \arrow[r, "\mathrm{B}"'] & h_6
\end{tikzcd}$$

Figure 2.2: Sequence of HUNL states and actions. Player two actions are capitalised. Call and Bet have been abbreviated as c and b, respectively.

For example, consider the sequence of HUNL events in Figure 2.2. Given the definitions above, we get the following relations between probabilities

$$\begin{aligned}
\pi^\sigma(h_6) &= \sigma_c(\varnothing, 3\clubsuit\mathrm{A}\clubsuit/7\spadesuit\mathrm{Q}\heartsuit)\sigma_1(h_1,\mathrm{c})\sigma_2(h_2,\mathrm{B}) \\
&\quad * \sigma_1(h_3,\mathrm{c})\sigma_c(h_4, 2\clubsuit 4\diamondsuit\mathrm{Q}\clubsuit)\sigma_2(h_5,\mathrm{B}) \\
\pi^\sigma_c(h_6) &= \sigma_c(\varnothing, 3\clubsuit\mathrm{A}\clubsuit/7\spadesuit\mathrm{Q}\heartsuit)\sigma_c(h_4, 2\clubsuit 4\diamondsuit\mathrm{Q}\clubsuit) \\
\pi^\sigma_1(h_6) &= \sigma_1(h_1,\mathrm{c})\sigma_1(h_3,\mathrm{c}) \\
\pi^\sigma_2(h_6) &= \sigma_2(h_2,\mathrm{B})\sigma_2(h_5,\mathrm{B}) \\
\pi^\sigma_{-1}(h_6) &= \sigma_c(\varnothing, 3\clubsuit\mathrm{A}\clubsuit/7\spadesuit\mathrm{Q}\heartsuit)\sigma_2(h_2,\mathrm{B})\sigma_c(h_4, 2\clubsuit 4\diamondsuit\mathrm{Q}\clubsuit)\sigma_2(h_5,\mathrm{B}) \\
\pi^\sigma_{-2}(h_6) &= \sigma_c(\varnothing, 3\clubsuit\mathrm{A}\clubsuit/7\spadesuit\mathrm{Q}\heartsuit)\sigma_1(h_1,\mathrm{c})\sigma_1(h_3,\mathrm{c})\sigma_c(h_4, 2\clubsuit 4\diamondsuit\mathrm{Q}\clubsuit) \\
\pi^\sigma(h_6) &= \pi^\sigma_c(h_6)\pi^\sigma_1(h_6)\pi^\sigma_2(h_6) \\
\pi^\sigma(h_6) &= \pi^\sigma_1(h_6)\pi^\sigma_{-1}(h_6) \\
\pi^\sigma(h_3) &= \sigma_c(\varnothing, 3\clubsuit\mathrm{A}\clubsuit/7\spadesuit\mathrm{Q}\heartsuit)\sigma_1(h_1,\mathrm{c})\sigma_2(h_2,\mathrm{B}) \\
\pi^\sigma(h_6|h_3) &= \sigma_1(h_3,\mathrm{c})\sigma_c(h_4, 2\clubsuit 4\diamondsuit\mathrm{Q}\clubsuit)\sigma_2(h_5,\mathrm{B}) \\
\pi^\sigma_1(h_6|h_3) &= \sigma_1(h_3,\mathrm{c}) \\
\pi^\sigma(h_6) &= \pi^\sigma(h_3)\pi^\sigma(h_6|h_3)
\end{aligned}$$

2.4.2 Strategy Values

Given a strategy profile σ, player p's expected value of a non-terminal state h is $u^\sigma_p(h) = \sum_{z\in Z, h\sqsubset z} \pi^\sigma(z)u_p(z)$. The player p value of a strategy profile is $u^\sigma_p = u^\sigma_p(\varnothing)$ and the value of an information set $I \in \mathcal{I}_p$ is $u^\sigma_p(I) = \sum_{h\in I} u^\sigma_p(h)$. Note that $u^\sigma_p(h)$ and $u^\sigma_p(I)$ are not conditional values, and include the probability $\pi^\sigma(h)$ of reaching h or states $h \in I$.

My work makes frequent use of counterfactual values, introduced by Zinkevich *et al.* [94]. Counterfactual value differs from the standard expected value in that it does not consider the player's own probability of reaching the state (thus the use of "counterfactual" in the name: instead of following σ, what if the player had instead played to reach here?) The counterfactual value for player p of state h is $v_p^\sigma(h) = \sum_{z \in Z, h \sqsubset z} \pi_{-p}^\sigma(h)\pi^\sigma(z|h)u_p(z)$. Equivalently, we could say $v_p^\sigma(h) = \sum_{z \in Z, h \sqsubset z} \pi_p^\sigma(z|h)\pi_{-p}^\sigma(z)u_p(z)$. Extending the idea to information sets, the counterfactual value for player p of an information set $I \in \mathcal{I}_p$ is $v_p^\sigma(I) = \sum_{h \in I} v_p^\sigma(h)$. Similarly, $v_p^\sigma(I \cdot a) = \sum_{h \in I \cdot a} v_p^\sigma(h)$.

Note the expected value of a state can be expressed as the product of the counterfactual value of the state and the probability of reaching it, so that $u_p^\sigma(h) = \pi_p^\sigma(h)v_p^\sigma(h)$. With perfect recall, this also holds for information sets. This relationship also means $v_p^\sigma := v_p^\sigma(\varnothing) = u_p^\sigma(\varnothing)$ because the probability $\pi_p^\sigma(\varnothing)$ of player p reaching the beginning of the game must always be 1.

2.4.3 Best Response

If we have a strategy profile σ, a player p best response is a strategy for p that maximises their expected value if the other players play according to σ. That is, $\mathrm{br}_p(\sigma) = \mathrm{argmax}_{\sigma^* \in \Sigma_p} u_p^{\langle \sigma_{-p}, \sigma^* \rangle}$. If we know our opponent's strategy, playing a best response is said to be rational play. We can similarly define the best response value, $\mathrm{vbr}_p^\sigma = \max_{\sigma^* \in \Sigma_p} u_p^{\langle \sigma_{-p}, \sigma^* \rangle} = u_p^{\langle \sigma_{-p}, br_p(\sigma) \rangle}$.

A best response strategy is not necessarily unique, because there may be information sets where there are multiple actions which have the same value. In two-player, zero-sum games an opponent best response is a worst-case strategy for us, and minimises our expected value.

We can compute a best response value or strategy recursively, using counterfactual values. Roughly speaking, at some information set I, a player p's best response (counterfactual) value after taking an action a is the sum of best response values across all player p child information sets that can be reached by making action a. The best response value at I is the maximum of the values across all a. A strategy can be recovered from values by choosing the maximising action at all information sets.

We need to define a counterfactual version of the best response value: $\mathrm{cvbr}_p^\sigma = \max_{\sigma^* \in \Sigma_p} v_p^{\langle \sigma_{-p}, \sigma^* \rangle}$. As $u_p^\sigma = v_p^\sigma$, we have $\mathrm{vbr}_p^\sigma = \mathrm{cvbr}_p^\sigma$. As with all other values, we could also consider the best response counterfactual value of information sets $\text{cvbr}^\sigma_p(I) = \max_{\sigma^* \in \Sigma_p} v_p^{\langle \sigma_{-p}, \sigma^* \rangle}(I)$, with similar definitions for states, or actions after information sets.

We also need to define the children of information set I after action a: $C(I \cdot a) = \{J \in \mathcal{I}_{p(I)} | I \cdot a \sqsubset J, \nexists K \sqsubset J \text{ s.t. } I \cdot a \sqsubset K\}$, where $I \sqsubset J$ if and only if $\exists h \in I, j \in J$ such that $h \sqsubset j$, and $I \cdot a \sqsubset J$ defined similarly.

We can now describe the values recursively for $I \in \mathcal{I}_p$ and $a \in A(I)$ as $\text{cvbr}^\sigma_p(I \cdot a) = \sum_{I' \in \text{C}(I \cdot a)} \text{cvbr}^\sigma_p(I')$, and $\text{cvbr}^\sigma_p(I) = \max_{a \in A(I)} \text{cvbr}^\sigma_p(I \cdot a)$.

2.4.4 Optimal Play and Nash Equilibrium

A Nash equilibrium is a strategy profile where every player's strategy is a best response to the strategy profile. It is guaranteed to exist for any extensive-form game [67][5], although it may not be unique. Because every player is acting rationally in the sense that they are maximising their utility against their opponents, the Nash equilibrium is the basic solution concept describing optimal play in imperfect information games. I will say (exactly) solved to mean we have found a Nash equilibrium profile, and will say a player has (exactly) optimal play to mean they are following their strategy from a Nash equilibrium profile.

In the two player, zero-sum, perfect-recall games that are used in this work, a Nash equilibrium is equivalent to a solution to the max-min optimisation problem $\max_{\sigma_p \in \Sigma_p} \min_{\sigma_q \in \Sigma_q} v_p(\langle \sigma_p, \sigma_q \rangle)$. While the Nash equilibrium might not be unique, every equilibrium profile has the same player p value. This unique value is called the game value for player p, and playing σ_p from any equilibrium σ guarantees that p gets at least the game value against any opponent strategy, including a perfect (best-response) opponent.

A player strategy σ in a Nash equilibrium profile does not necessarily make rational (best-response) actions in all parts of the game. At any information set $I \in \mathcal{I}_p$ that is not reached by σ_{-p} (*i.e.*, $\pi_{-p}(h) = 0 \ \forall h \in I$) a Nash equilibrium only has a guarantee that σ_p gets at least the game value against an opponent strategy which does reach I, rather than guaranteeing that σ_p

[5] Existence is only guaranteed when using mixed strategies, described in Section 2.5.1, or in perfect-recall games where the common strategy spaces are equivalent.

makes rational decisions after I. Although I do not consider them in this work, there are refinements of the Nash equilibrium solution concept, which address this potentially irrational play in these unreached situations. For example, the sequential equilibrium [52], the (normal form) trembling hand equilibrium [77], and the quasi-perfect equilibrium [89] guarantee rational play.

There are additional issues with the Nash equilibrium in games which are not constant-sum, or have more than two players. Max-min, min-max, and Nash equilibrium strategy profiles are no longer equivalent, and there is no unique game value. Nash equilibrium play might not seem sensible, as seen in the traveler's dilemma game [3]. Equilibrium strategies are not interchangeable, so a player's strategy might no longer be a best response if their opponents are playing according to a different equilibrium. A Nash equilibrium is also hard to compute (or even approximately compute) if the game has more than two players [21], is not constant-sum [19], or is not perfect recall [50].

2.4.5 Approximately Optimal Play

Instead of exactly computing a Nash equilibrium, we are often satisfied with an approximate solution, such as a strategy produced by an iterative algorithm which produces successively better strategies. I will generally drop the word approximate when talking about an approximate solution, and will use solved to mean we have found a sufficiently good approximation of a Nash equilibrium. I will use the term exactly solved or exact equilibrium profile to distinguish the case where we have found a Nash equilibrium rather than an approximation of a Nash equilibrium.

An ϵ-Nash equilibrium is an approximate solution, where for each player p, $\mathrm{vbr}_p^\sigma - u_p^\sigma \le \epsilon$. I will use exploitability of a strategy profile to mean the average of the players' best response values: $(1/2)\sum_{p\in P} \mathrm{vbr}_p^\sigma$. An ϵ-Nash has at most ϵ exploitability, and a profile with ϵ exploitability is a 2ϵ-Nash equilibrium. A profile has 0 exploitability if and only if it is an exact Nash equilibrium.

I will use exploitability of a single player p's strategy σ_p to mean the difference between the game value g_p and the worst-case value: $g_p - u_p^{\langle \sigma_p, \mathrm{br}_{-p}(\sigma_p)\rangle}$. There is a connection to strategy profile exploitability: if both players have a strategy with ϵ exploitability, the strategy profile has ϵ exploitability.

With Bowling *et al.*, I introduced the notion of essentially solved [9] to answer the question of when an approximation is good enough. We say a game is essentially solved if we have an approximate equilibrium that cannot be distinguished from an exact equilibrium with statistical confidence after observing a human lifetime of play. The minimum exploitability needed to essentially solve a game depends on both the speed of human play, and the variance of game outcomes.

2.5 Strategy Spaces

There are a number of different ways to express a strategy, including pure strategies, mixed strategies, behaviour strategies, and sequence form strategies. Somewhat surprisingly these different strategy spaces are not entirely equivalent[6] in general games, although they are equivalent in perfect-recall games. Unless explicitly noted, I will be using behaviour strategies throughout this thesis.

2.5.1 Normal Form and Mixed Strategies

The earliest methods of solving a two player, zero-sum game [23, 20, 88] involved translating it into a different description, called a normal form game. Given a game, for each player we construct a set of pure strategies. Each player p pure strategy is a deterministic policy for p, mapping each information set $I \in \mathcal{I}_p$ to a single action $a \in A(I)$. The normal form of a game is a payoff matrix indexed by the sets of all pure strategies for each player, with entries that are the expected value of the row and column strategies.

Non-determinism in player actions is introduced by using a mixed strategy, which is a probability distribution over the set of the player's pure strategies. Playing a mixed strategy means sampling a pure strategy from this distribution at the beginning of the game, then using that pure strategy to select actions

[6]Any strategy profile induces a probability distribution of reaching each terminal state. A set of strategies are equivalent if the set of terminal probability distributions are identical.

at each information set.

In normal form, the constraints on the strategy space are linear (the probabilities must sum to 1) and the expected value of a mixed strategy profile $\langle \sigma_p, \sigma_q \rangle$ is a bilinear function $\sigma_p^\intercal U \sigma_q$ of the strategies and some payoff matrix U. The problem of finding a mixed strategy Nash equilibrium can be expressed as a linear program (LP), giving us an algorithm for solving an extensive form game: translate it to normal form, generate and solve the appropriate linear program, and use the resulting mixed strategy. While solving an LP is polynomial [49], the first translation step is not. The number of possible pure strategies is exponential in the number of information sets.

2.5.2 Sequence Form

Romanovskii [75] and later Koller *et al.* [51] introduced sequence form games, which do not have the exponential blowup of normal form games, and gave a polynomial time algorithm for solving two player, zero-sum, perfect-recall extensive form games. Here, a player p sequence refers to an information set $I \in \mathcal{I}_p$ and action $a \in A(I)$, and the sequence of player p information sets and actions required to reach I. In perfect recall games, the history to reach I is unique, and every sequence can be represented just by I and a.

Sequences in sequence form play the role of pure strategies in normal form. Like pure strategies, given a sequence for each player, there is an associated expected value. Note that many tuples of sequences are not compatible and have a value of 0, because a sequence might require the opponent to make a move that does not match the player's sequence. For example, a row sequence might correspond to a choice after the opponent calls, but the column sequence corresponds to the opponent folding. Unlike pure strategies, the number of player p sequences is smaller than the number of game states, and is equal to $1 + \sum_{I \in \mathcal{I}_p} |A(I)|$.

Like mixed strategies and pure strategies, a sequence form strategy is defined in terms of sequences, but is not a single probability distribution. Instead, it specifies $\pi_p^\sigma(I, a)$ for each sequence, which is a set of probability distributions that retains something of the tree-like structure of the extensive form game. In a sequence form strategy, the probability of a parent sequence[7] must be equal to the sum of the probabilities of the child sequences. To play a sequence form strategy, we can normalise the sequence probabilities to get $\sigma(I, a)$.

Continuing the parallels between normal form and sequence form, the payoff of a sequence form strategy profile $\langle \sigma_p, \sigma_q \rangle$ is again a bilinear function $\sigma_p^\intercal U \sigma_q$ of some payoff matrix U. Koller *et al.* showed that the set of sequence form strategies can be described by linear constraints, that a sequence form Nash equilibrium must exist in two player, zero-sum, perfect-recall games, and gave a linear program for finding a sequence form Nash equilibrium [51].

2.5.3 Behaviour Strategies

A behaviour strategy σ_p directly specifies the probabilities $\sigma_p(I, a)$. When playing a game according to a behaviour strategy, at every information set I the player samples from the distribution $\sigma(I)$ to get an action. Similar to sequence form strategies, the size of a behaviour strategy is $\sum_{I \in \mathcal{I}_p} |A(I)|$. The first down side of behaviour strategies is that the expected value is not linear in the strategy probabilities, or even a convex-concave function.

The second problem is that computing the average of a behaviour strategy is not a matter of computing the average of the specified action probabilities: it is not entirely reasonable to give full weight to a strategy's action probabilities for an information set that is reached infrequently. Instead, to compute the average behaviour strategy we must weight $\sigma_p(I, a)$ by $\pi_p^\sigma(I)$, which is equivalent to averaging the equivalent sequence form strategies.

2.6 Abstraction

We may want to use an approximation of a Nash equilibrium to play a game, but many games are too large to solve. For example, even using some game-specific knowledge to eliminate identical situations, storing a complete HULHE strategy would require 131 TB using 4 byte floating point numbers [9], and would require a correspondingly large computation time. While Chapter 4

[7]The parent of a sequence $\langle I_1, a_1 \rangle, ..., \langle I_n, a_n \rangle$ is $\langle I_1, a_1 \rangle, ..., \langle I_{n-1}, a_{n-1} \rangle$.

discusses a method we used to solve HULHE, finding a solution to the game had previously been intractable.

Shi and Littman suggested the use of abstraction [79] as one method used to generate a strategy for very large games. By generating an abstraction of the game and trying to capture the general form of the rules, we can create an abstract game which is a small model of the original game. If the abstract game is small enough, we can solve it, and then use the abstract strategy to guide our play in the original game. Assuming our abstract game has captured the strategically important properties of the original game, our abstract strategy is hopefully a good policy in the original game.

Abstraction has been popular in game-theoretic poker research, and one method of generating abstract poker games has involved grouping information sets together into a collection called a bin or a bucket [79, 4, 32]. The quality of the strategy depends on the similarity of the grouped situations [57], and there has been a series of increasingly complicated algorithms for automatically generating abstract poker games [33, 29, 13]. Grouping different card situations together into buckets has been sufficient to generate strong HULHE strategies [46], and was used to beat top human specialists [10].

It is easy to translate the abstract equilibrium strategy from a bucket-based abstract game back into the original game. There was some mapping function used to map an information set into a bucket when creating the game, and we use this same mapping function to map an observed information set into an abstract information set. Given the abstract information set, we directly use the action probabilities from the abstract strategy.

The game of HUNL is much larger, and methods for generating an abstract game for HUNL have generally involved a combination of bucket-based abstraction for card information, as well as drastically pruning the game by using a sparse subset of available player actions. Pruning the available actions in the abstract game sets up a non-trivial problem in using the abstract strategy [76, 27]. If the opponent makes an action that we pruned away, there is no longer an obvious way to play after that action was made, because there is no obvious mapping from the resulting state to a state in the abstract game.

We might hope for some guarantee of improvement by using a larger, better informed abstraction, and this has generally been the case with poker games. We have observed that, as we use finer-grained abstraction and better mapping functions, one on one performance increases and exploitability decreases [46]. There are, however, no general theoretical guarantees. It is possible to see pathological behaviour where the exploitability of an abstract strategy increases when using a larger abstraction, even if the larger abstraction is a strict refinement of the smaller abstraction [92]. It requires great care, and an expensive computation, to generate an abstract game with any sort of a-priori bounds on the exploitability within the original game [53].

2.7 Regret

Much of my work makes use of the mathematical concept of regret. In ordinary use, regret is a feeling that we would rather have done something else. The mathematical notion of regret tries to capture the colloquial sense of regret, and is a hindsight measurement of how much better we could have done compared to how we did do. Assume we have a fixed set of actions A, a sequence of probability distributions σ^t over A, and a sequence of value functions $v^t : A \mapsto \mathbb{R}$[8].Given some set of alternative strategies S, where $s \in S$ maps time $t \in [1...T]$ to a probability distribution $s(t)$ over A, our regret at time T with respect to S is $R^T_S = \max_{s \in S} \sum_t s(t) \cdot v^t - \sum_t \sigma^t \cdot v^t$.

In order for regret to be meaningful, we need bounded values, so that there is some L that $|v^t(a) - v^t(b)| \leq L\ \forall t$ and $\forall a, b \in A$.

Using different sets S gives different regret measures. The most common, external regret, uses S such that $\forall s \in S$, $s(t)$ is a probability distribution placing all weight on a single action A, and $\forall t, t'\ s(t) = s(t')$. That is, external regret considers how we would have done if we had always played a single, fixed action instead. Unless specified otherwise, I will use regret to refer to external regret.

I will often speak of the regret $R^T(s)$ of a choice s, by which I will mean

[8]While A is fixed, both σ^t and v^t are free to vary over time.

$R^T(s) = \sum_t s(t) \cdot v^t - \sum_t \sigma^t \cdot v^t$. The regret can then be written in terms of the choice regrets as $R^T_S = \max_{s \in S} R^T(s)$. For external regret, this idea can be simplified to the regret for an action $a \in A$.

2.7.1 Online Learning and Regret Minimisation

Let us say we have a repeated, online, decision making problem with a fixed set of actions. At each time step, we must choose an action without knowing the values in advance. After making an action we receive some value (also known as reward), and also get to observe the values for all other actions. The values are arbitrary, not fixed or stochastic, and might be chosen in an adversarial fashion. The adversary has the power to look at our decision-making rule for the current time step before we make the action, but if we have a randomised policy the adversary does not have the power to determine the private randomness used to sample from the distribution. This setting is often called expert selection, where the actions can be thought of as experts, and we are trying to minimise loss rather than maximise value.

Given such a problem, with arbitrary adversarial values, regret is a natural measure of performance. Looking at our accumulated value by itself has little meaning, because it might be low, but could still be almost as high as it would have been with any other policy for selecting actions. So, we would like to have an algorithm for selecting actions that guarantees we have low regret. Because of the bounds on values, we can never do worse than LT regret. Cesa-Bianchi *et al.* give a lower bound: for any $\epsilon > 0$ and sufficiently large T and $|A|$, any algorithm has at least $(1-\epsilon)L\sqrt{T/2\ln(|A|)}$ external regret in the worst case [17].

We are often interested in the average behaviour, or the behaviour in the limit, and so we might consider dividing the total regret by T to get the average regret. If our total regret is sub-linear, average regret will approach 0 as $T \to \infty$, and our average value (or reward) approaches the best-case value. Despite the arbitrary, potentially adversarial selection of values at each time step, there are multiple algorithms which guarantee sub-linear external regret.

2.7.2 Regret-Matching Algorithm

Blackwell introduced the Regret-matching algorithm [5], which can be used[9] to solve the online learning problem discussed in Section 2.7.1. Given the regrets $R^T(a)$ for each action, the regret-matching algorithm specifies a strategy $\sigma^T(a) = R^{T-1}(a)^+ / \sum_{b \in A} R^{T-1}(b)^+$, where $x^+ = \max(x, 0)$. That is, the current strategy is the normalised positive regrets.

Rather than computing $R^T(a)$, an implementation of regret-matching will usually maintain a single regret value for each action, and incrementally update the regrets after each observation. We can rewrite the action regrets as $R^T(a) = R^{T-1}(a) + \Delta R^T(a)$, where $\Delta R^t(a) = v^t(a) - \sum_{b \in A} \sigma^t(b) v^t(b)$ and $R^0(a) = 0$. Using $\Delta R^T(a)$, which depends only on the newly observed values v^T, we can update the previous regrets $R^{T-1}(a)$ to get $R^T(a)$.

Given T steps and a bound L^t on $|v^t(a) - v^t(b)|$, regret-matching has sub-linear regret, with an upper bound $R^T \leq \sqrt{\sum_{t=1}^{T} (L^t)^2 |A|}$. With a single fixed value bound L, the regret bound can be simplified to $R^T \leq L\sqrt{T|A|}$. Regret-matching has two other desirable properties: it is simple, because the current strategy is just the normalised positive regrets, and it has no parameters which need to be tuned.

2.7.3 Hedge Algorithm

Based on the weighted majority algorithm of Littlestone *et al.* [61], Freund and Schapire introduced the Hedge [26] algorithm to solve a variant of the online learning problem[10] of Section 2.7.1. Given a current strategy σ^{t-1} and observed values v^t, Hedge(β) uses a new strategy $\sigma^t = \sigma^{t-1} \beta^{v^t} / \sum_a \beta^{v^t(a)}$. The initial strategy σ^0 is a uniform random distribution over A.

The Hedge algorithm can be written in a different, mathematically equivalent way by storing action values $V^T(a) = \sum_{t=1}^{T} v^t(a)$ and using a strategy

[9]Blackwell used the algorithm as part of a constructive proof for an argument about optimising multi-valued objectives. The application to minimising regret is a specific application of the argument, using action regrets as the multiple objectives. I introduce the algorithm here, as it is a component of the CFR game solving algorithm introduced in Chapter 3.

[10]The original setting considered experts which predict a value at each time, with the loss being the difference from the observed value.

$\sigma^t = e^{\gamma V^{t-1}} / \sum_a e^{\gamma V^{t-1}(a)}$. Hedge is shift-invariant, where adding a constant to all values results in the same strategy, so we could also write the algorithm as storing action regrets $R^t(a)$ and using a strategy $\sigma^t = e^{\gamma R^{t-1}} / \sum_a e^{\gamma R^{t-1}(a)}$.

If we know ahead of time the total number of time steps T we will run, and use $\beta = 1 + \sqrt{2\ln(|A|)/T}$, Hedge($\beta$) has sub-linear regret with an upper bound $R^T \leq L\sqrt{2T\ln(|A|)} + \ln(|A|)$ [26]. If we do not know T, we can use a doubling trick where we tune β for k steps, then $2k$ steps, then $4k$ steps and so on. The asymptotic complexity remains the same.

Hedge is overwhelmingly more popular than regret-matching, due to its $\sqrt{\ln(|A|)}$ dependence on the number of actions, compared to the $\sqrt{|A|}$ behaviour of regret-matching. The parameterised nature of Hedge, however, is a mixed blessing. In practice, we can drop the theoretically justified choice of β and get significantly better performance by tuning the choice of β for the problem. The promise of much better performance means there is now a parameter to tweak, in order to chase that performance. Other algorithms, like NormalHedge [18] and AdaNormalHedge [62], have the same $\sqrt{\ln(|A|)}$ dependence without the free parameter.

2.7.4 Regret and Exploitability

Regret minimisation is relevant to game solving, because there is a connection between regret and exploitability. If we have a sequence of strategy profiles, we can say the player p regret is $R_p^T = \max_{\sigma^*} \sum_{t=1}^T v_p^{\langle \sigma^*, \sigma_{-p}^t \rangle} - \sum_{t=1}^T v_p^{\sigma^t}$. The max term is T times the best response value to the average opponent strategy $\bar{\sigma}_{-p}$, so the exploitability of the average strategy profile $\bar{\sigma}$ is $(1/2)T(R_1^T + R_2^T)$.

The connection above means that we can solve games through self-play of regret minimising algorithms. If each player minimises external regret over the set of pure strategies, the profile of average strategies is a $1/\sqrt{T}$-Nash equilibrium. Unfortunately, this is an impractical algorithm because the number of pure strategies is exponential in the size of the extensive-form game.

2.8 Setting for this Work

All of the theory in the work applies to two player, zero-sum, perfect-recall, imperfect information games, described as an extensive form game. I have used HULHE and HUNL as the full-size problems for experimental tests.

Chapter 3

CFR: State of the Art

Counterfactual regret minimisation (CFR) is an algorithm for finding an ϵ-Nash equilibrium in two player zero-sum games, developed by Zinkevich *et al.* [94] in response to the high memory cost of earlier algorithms. CFR uses memory on the order of the strategy size, rather than the game size. It is necessary to examine CFR in detail, as the majority of my contributions extend the CFR algorithm in some way.

My main contribution related to CFR is a tighter regret bound (not previously published). The bound is a strict improvement on the current best known bound, and shows that CFR does not always pay a large performance penalty for games with a large chance branching factor. I was also involved on earlier shared work on a CFR regret bound (published in NIPS [30].)

3.1 CFR Algorithm

CFR is a self-play algorithm using regret minimisation. Zinkevich *et al.* introduced the idea of counterfactual value, a new utility function on states and information sets, and use this value to independently minimise regret at every information set. By using many small regret minimisation problems, CFR overcomes the prohibitive memory cost of directly using a regret minimisation algorithm over the space of pure strategies.

Counterfactual values, defined in Section 2, can be combined with any standard regret concept to get a counterfactual regret at an information set. CFR uses external regret, so that the counterfactual regret at time T of an action a at a player p information set I is

$$R^T(I,a) = \sum_{t=1}^{T} v_p^{\sigma^t}(I \cdot a) - \sum_{t=1}^{T} \sum_{a \in A(I)} \sigma^t(I,a) v_p^{\sigma^t}(I,a)$$

Given the sequence of past strategies and values for I, any regret-minimising algorithm can be used to generate a new strategy $\sigma_p^t(I,a)$ over the actions $A(I)$, with a sub-linear bound on total external regret $R^T(I) = \max_{a \in A(I)} R^T(I,a)$. CFR uses the regret-matching algorithm. Combining $\sigma_p^t(I,a)$ at each player p information set gives us a complete behaviour strategy σ_p^t, and repeating the process for all players gives us a strategy profile σ^t.

The CFR algorithm proceeds in an iterative fashion. At each iteration, there are four conceptual steps:

1. Generate strategy profile σ^t from the regrets, as described above. For all $I \in \mathcal{I}$, $a \in A(I)$, and $p = p(I)$:

$$\sigma_p^t(I,a) = \begin{cases} R^t(I,a)^+ / \sum_{b \in A(I)} R^t(I,b)^+ & \sum_{b \in A(I)} R^t(I,b)^+ > 0 \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases}$$

2. Update the average strategy profile to include the new strategy profile. For all $I \in \mathcal{I}$, $a \in A(I)$, and $p = p(I)$:

$$\begin{aligned} \bar{\sigma}_p^t(I,a) &= \frac{1}{t} \sum_{t'=1}^{t} \pi_p^{\sigma^t}(I) \sigma_p^{t'}(I,a) \\ &= \frac{t-1}{t} \bar{\sigma}_p^{t-1}(I,a) + \frac{1}{t} \pi_p^{\sigma^t}(I) \sigma_p^t(I,a) \end{aligned}$$

3. Using the new strategy profile, compute counterfactual values. For all $I \in \mathcal{I}$, $a \in A(I)$, and $p = p(I)$:

$$\begin{aligned} v_p^{\sigma^t}(I,a) &= \sum_{h \in I \cdot a} v_p^{\sigma^t}(h) \\ &= \sum_{h \in I \cdot a} \sum_{z \in Z, h \sqsubset z} \pi_{-p}^{\sigma^t}(h) \pi^{\sigma^t}(z|h) u_p(z) \end{aligned}$$

4. Update the regrets using the new counterfactual values. For all $I \in \mathcal{I}$, $a \in A(I)$, and $p = p(I)$:

$$R^t(I,a) = R^{t-1}(I,a) + v_p^{\sigma^t}(I,a) - \sum_{a \in A(I)} \sigma^t(I,a) v_p^{\sigma^t}(I,a)$$

The first and fourth steps are specific to the choice of regret-matching as an online regret minimisation algorithm, and could be replaced with any other regret minimising algorithm.

A general version of CFR with no performance improvements is given in Algorithm 1. R and $\bar{\sigma}$ are tables which store the regrets and average strategy, respectively, for each information set I and action a. σ and v are temporary storage tables which store the current strategy and counterfactual values, respectively. We ignore any player subscripts in the algorithm, as the information set used as part of the table index is sufficient to determine the acting player. For example, Algorithm 1 uses the table entry $v(I, a)$ to refer to the counterfactual value $v_{p(I)}(I, a)$.

It is important to note that while CFR deals with behaviour strategies (separate probability distributions across actions at each information set), the average must still be done in the space of mixed strategies or sequence form strategies, as noted in Section 2.5.3. Using the more compact sequence form representation, maintaining the average strategy for player p can be done by incrementally updating $\bar{\sigma}_p(I, a) = \sum_{t=1}^{T} \pi_p^t(I)\sigma^t(I, a)$ for each information set $I \in \mathcal{I}_p$ and action $a \in A(I)$. We can ignore the factor to turn the sum into an average, because the process of turning a sequence form strategy into a behaviour strategy involves normalising the probabilities at each information set, canceling out the missing constant.

An implementation of CFR can be made more efficient by combining the strategy computation, average strategy update, and value computation in a depth-first traversal of states in the game tree. A version of CFR which does this recursive traversal is given in Algorithm 2. While it still uses temporary space $v(I, a)$ for every information set I and action a, Algorithm 2 does not store a complete copy of the current strategy σ. At each node h, sequence probabilities $\pi_p(h)$ are passed in for both players, and values from the children are used to compute the counterfactual values $v_p(h)$ for both players, which are returned to the parent.

There are also a number of other game-specific implementation details which are important for good performance, as mentioned by Johanson [43].

```
input  : Game $\mathcal{G}$, Number of iterations $T$
output: $\epsilon$-Nash approximation $\bar{\sigma}$
foreach $I \in \mathcal{I}$ do
    foreach $a \in A(I)$ do
        $R(I,a) \leftarrow 0$
        $\bar{\sigma}(I,a) \leftarrow 0$
for $t \leftarrow 1$ to $T$ do
    $\sigma \leftarrow$RegretToStrategy($\mathcal{I}$,$R$)
    UpdateAverage($\mathcal{I}$,$\sigma$,1,$\bar{\sigma}$)
    $v \leftarrow$ StrategyToValues($\mathcal{I}$,$\sigma$)
    UpdateRegrets($\mathcal{I}$,$\sigma$,$v$,$R$)
NormaliseAverage($\mathcal{I}$,$\bar{\sigma}$)
return $\bar{\sigma}$

function RegretToStrategy($UpdateSets$,$R$):
    foreach $I \in UpdateSets$ do
        $\sigma(I) \leftarrow$ RegretMatchingStrategy($I$,$R$)
    return $\sigma$

function RegretMatchingStrategy($I$):
    $sum \leftarrow \sum_{a \in A(I)} R(I,a)^+$
    if $sum > 0$ then
        foreach $a \in A(I)$ do
            $s(a) \leftarrow R(I,a)^+/sum$
    else
        foreach $a \in A(I)$ do
            $s(a) \leftarrow 1/|A(I)|$
    return $s$

function UpdateAverage($UpdateSets$,$\sigma$,$w$,$\bar{\sigma}$):
    foreach $I \in UpdateSets, a \in A(I)$ do
        $\bar{\sigma}(I,a) \leftarrow \bar{\sigma}(I,a) + w\pi_{p(I)}(I)\sigma(I,a)$

function StrategyToValues($UpdateSets$,$\sigma$):
    foreach $I \in UpdateSets, a \in A(I)$ do
        $v(I,a) \leftarrow \sum_{h \in I \cdot a} \sum_{z \in Z, h \sqsubset z} \pi_{-p(I)}(h)\pi(z|h)u_{p(I)}(z)$
    return $v$

function UpdateRegrets($UpdateSets$,$\sigma$,$v$,$R$):
    foreach $I \in UpdateSets, a \in A(I)$ do
        $R(I,a) \leftarrow R(I,a) + v(I,a) - \sum_{b \in A(I)} \sigma(I,b)v(I,b)$

function NormaliseAverage($UpdateSets$,$\bar{\sigma}$):
    foreach $I \in UpdateSets$ do
        $sum \leftarrow \sum_{a \in A(I)} \sigma(\bar{I},a)$
        foreach $a \in A(I)$ do
            $\bar{\sigma}(I,a) \leftarrow \bar{\sigma}(I,a)/sum$
```

Algorithm 1: CFR

```
input  : Game $\mathcal{G}$, Number of iterations $T$
output: $\epsilon$-Nash approximation $\bar{\sigma}$
foreach $I \in \mathcal{I}$ do
    foreach $a \in A(I)$ do
        $R(I, a) \leftarrow 0$
        $\bar{\sigma}(I, a) \leftarrow 0$
for $t \leftarrow 1$ to $T$ do
    foreach $p \in \{p_1, p_2\}$ do
        $s[p] \leftarrow 1$
    RecursiveUpdate($\varnothing$, $s[\cdot]$)
    UpdateRegrets($\mathcal{I}$,$\sigma$,$v$,$R$)
NormaliseAverage($\mathcal{I}$,$\bar{\sigma}$)
return $\bar{\sigma}$

function RecursiveUpdate($h$,$s[\cdot]$):
    input  : State and sequence probabilities $\pi_p(h)$ for all $p$
    output: Counterfactual values for all $p$
    if $h \in Z$ then
        return $[s[p_2]\pi_c(h)u_{p_1}(h), s[p_1]\pi_c(h)u_{p_2}(h)]$
    $I \leftarrow I \in \mathcal{I}$ such that $h \in I$
    $r \leftarrow [0, 0]$
    if $p(h) = c$ then
        foreach $a \in A(h)$ do
            $r \leftarrow r$ + RecursiveUpdate($h \cdot a$, $s$)
    else
        $s' \leftarrow s$
        $opp \leftarrow p \in P$ such that $p \neq p(h)$
        $\sigma \leftarrow$ RegretMatchingStrategy($I$)
        foreach $a \in A(h)$ do
            $s'[p(h)] \leftarrow s[p(h)] * \sigma[a]$
            $r' \leftarrow$ RecursiveUpdate($h \cdot a$, $s'$)
            $v(I, a) \leftarrow v(I, a) + r'[p(h)]$
            $r[p(h)] \leftarrow r[p(h)] + r'[p(h)]\sigma[a]$
            $r[opp] \leftarrow r[opp] + r'[opp]$
    return $r$
```

Algorithm 2: Recursive Implementation of CFR

For example, poker-like games can be split into a small amount of private information (the private cards in a player's hand) and a tree of publicly visible actions and chance events. That is, a player's information set is the public state plus their private information. In this case, we can further extend the ideas of Algorithm 2 and perform an entire iteration in a single depth-first traversal. The recursive update now operates on public tree nodes t, takes as an input a vector of sequence probabilities across the private information for each player, and returns a vector of counterfactual values across the private information for each player. Algorithm 3 gives a sketch of this style of game-specific implementation for public trees, which no longer uses either v or σ for temporary storage. Game specific details for handling chance events and terminal evaluation are omitted.

3.1.1 Regret Bounds

Zinkevich *et al.* showed that the player p regret over pure strategies is bounded by the sum of counterfactual regrets at all player p information sets [94].

$$R_p^T := \max_{\sigma_p^D \in \Sigma_p^D} \left(\sum_{t=1}^{T} u_p^{\langle \sigma_{-p}^t, \sigma_p^D \rangle} - \sum_{t=1}^{T} u_p^{\sigma^t} \right) \leq \sum_{I \in \mathcal{I}_p} R^T(I)$$

where Σ_p^D is the set of all player p pure strategies. Because we can alway find a value-maximising (best response) pure strategy, the regret over all strategies is equal to the regret over pure strategies.

If the maximum difference in leaf utilities is L and $A = \max_{I \in \mathcal{I}} |A(I)|$, then the bound on regret-matching performance gives us $R_p^T \leq L|\mathcal{I}_p|\sqrt{A_p T}$. Given the connection between regret for a sequence of strategies and exploitability of the average strategy, the average strategy profile $\bar{\sigma} = (1/T)\sum_{t=1}^{T} \sigma^t$ is an $L|\mathcal{I}|)\sqrt{A/T}$-Nash equilibrium.

The CFR regret bound and the associated exploitability bound can be broken into pieces. T is the running time, if we are only counting iterations rather than seconds. $|\mathcal{I}|$ is the number of information sets, which is one potential way to measure the size of an extensive form game[1]. A gives the maximum branch-

[1]It is also reasonable to look at the number of states $|H|$, and the number of strategy probabilities $|\{\langle I, a \rangle | I \in \mathcal{I}, a \in A(I)\}|$

```
input  : Game $\mathcal{G}$, Number of iterations $T$
output: $\epsilon$-Nash approximation $\bar{\sigma}$
foreach $I \in \mathcal{I}$ do
    foreach $a \in A(I)$ do
        $R(I, a) \leftarrow 0$
        $\bar{\sigma}(I, a) \leftarrow 0$
for $t \leftarrow 1$ to $T$ do
    foreach $p \in \{p_1, p_2\}, private_p$ do
        $s[p][private] \leftarrow 1$
    Update($\varnothing$, $s[\cdot][\cdot]$)
NormaliseAverage($\mathcal{I}$,$\bar{\sigma}$)
return $\bar{\sigma}$

function Update($t$,$s[\cdot][\cdot]$):
    input  : Public state and $\pi_p(t \cdot private)$ for all $p$ and $private_p$
    output: Counterfactual values for all $p$ and $private_p$
    if $h \in Z$ then
        foreach $p \in \{p_1, p_2\}, private_p$ do
            $r[p][private_p] \leftarrow v_p(t \cdot private_p)$
        return $r$
    foreach $p \in \{p_1, p_2\}, private_p$ do
        $r[p][private_p] \leftarrow 0$
    if $p(t) = c$ then
        foreach $a \in A(t)$ do
            $s' \leftarrow$ ChanceChildSequenceProbs($s$,$t$,$a$)
            $r' \leftarrow$ Update($t \cdot a$, $s'$)
            $r \leftarrow$ ChanceGatherValues($r$,$t$,$a$,$r'$)
    else
        $s' \leftarrow s$
        foreach $private_p$ do
            $\sigma[private_p] \leftarrow$ RegretMatchingStrategy($t \cdot private_p$)
        foreach $a \in A(t)$ do
            foreach $private_p$ do
                $s'[p(h)][private_p] \leftarrow s[p(h)][private_p] * \sigma[private_]][a]$
            $r' \leftarrow$ Update($t \cdot a$, $s'$)
            foreach $private_p$ do
                $v(t \cdot private_p, a) \leftarrow v(t \cdot private_p, a) + r'[p(h)][private_p]$
                $r[p(h)][private_p] \leftarrow$
                  $r[p(h)][private_p] + r'[p(h)][private_p]\sigma[private_p][a]$
            foreach $private_{opp}$ do
                $r[opp][private_{opp}] \leftarrow r[opp][private_{opp}] + r'[opp][private_{opp}]$
    return $r$
```

Algorithm 3: Sketch of a Game-Specific Implementation of CFR

ing factor of the game tree. L describes the range of terminal values, and can be ignored if we are willing to re-scale the problem so that all terminal values are in $[0, 1]$. So, the CFR regret bound of Zinkevich *et al.* shows that given an $\epsilon > 0$ we can find an ϵ-Nash equilibrium by running enough iterations, and for a fixed ϵ the maximum number of iterations we need grows roughly linearly in the size of the game.

Lanctot *et al.* give a tighter bound on the regret, showing better scaling in the size of the game. They showed that $R_p^T \leq LM_p\sqrt{AT}$, for a more complicated constant M_p that is smaller than $|\mathcal{I}_p|$ [58]. Let $\mathcal{B}_p$ be the partition of $\mathcal{I}_p$ such that for any $B \in \mathcal{B}_p$ and any $I, I' \in B$, player p made the same sequence of actions to reach both I and I'. Then $M_p = \sum_{B\in\mathcal{B}_p} \sqrt{B}$.

Note that the partition $\mathcal{B}_p$ is well defined because of perfect recall. As described in Section 2.3.2, all states in an information set have passed through the same sequence of player p information sets, and made the same action at those earlier information sets. Loosely speaking, M_p is roughly the size of a tree built out of player p actions (*i.e.,* $|\mathcal{B}|$), multiplied by the square root of the size of a tree built out of chance and opponent actions (*i.e.,* $\max_{B\in\mathcal{B}_p} \sqrt{B}$.)

Gibson *et al.* improve on the bound of Lanctot *et al.*, further improving the scaling with respect to game size. We showed that $R_p^T \leq LM_p\sqrt{AT}$ for a smaller M_p constant [30]. I was involved with this work, starting with the observation that the inequality of Zinkevich *et al.* in Theorem 3 [94] is an equality, described in Theorem 4 of [30]. Given the same partition $\mathcal{B}_p$ described above, let $M_p(\sigma_p^*) = \sum_{B\in\mathcal{B}_p} \max_{I\in B} \pi_p^{\sigma^*}(I)\sqrt{|B|}$ for any player p best response σ_p^* to $\bar{\sigma}$. Then $M_p = \max_{\sigma_p^*} M_p(\sigma_p^*)$.

Speaking loosely again, the improved M_p constant is roughly the depth of a tree built out of player p actions (*i.e.,* $\pi_p^{\sigma^*}(I)$ will only be non-zero for a single sequence of player p actions), multiplied by the square root of the size of a tree built out of chance and opponent actions. That is, we showed that CFR self-play regret, and therefore exploitability, grows as a depth-like quantity with respect to player actions, rather than a tree-size quantity.

3.2 Updated Regret Bound

We can extend the technique used by Gibson *et al.* to get an even tighter bound with respect to game size in many cases, and a significantly tighter bound for games with many sequences of chance events. For the game of HULHE, the bound using the work presented here is at least 26 000 times smaller than the bound achieved by previous work. This result is previously unpublished.

The previous M_p bounds work by partitioning $\mathcal{I}_p$ in such a way that we can show that the $\sum_{I \in B} \sqrt{\sum_{t=1}^{T} x_B^2} \leq \sqrt{|B|T}$ for all parts B. Gibson *et al.* used the same partition as Lanctot *et al.*, but improved the bounds by weighting each partition by the player's probability of reaching the information sets, and then considering (pure) best response player strategies. I improve the bounds by considering pure strategies earlier in the proof, and including the chance probability in the weighting for each partition.

The updated regret bound partitions $\mathcal{I}_p$ by the number of actions p took to reach the information set. In the general form, the bound requires an additional user-provided way to sub-partition each part. Section 3.2.2 discusses some special cases that give specific ways to create these sub-partitions. The intention of these sub-partitions is to group together states with different chance events, and as few as possible different sequences of opponent actions.

Informally, for games where the information sets at every depth can be exactly partitioned by chance events, the M_p constant of this section is roughly the depth of a tree built out of player p actions, multiplied by the square root of the size of a tree built out of opponent actions. For this special case, the new bound improves on the previous best bound by Gibson *et al.* by eliminating the dependence on chance events. More generally, the dependence on chance events is not eliminated, but can still be reduced (in a non-obvious way) from the bound of Gibson *et al.*.

3.2.1 Formal Description

The bound and its proof requires some additional terminology. The opponent of player p is opp(p). Let $\chi_{-p}^{\sigma}(I) = \sum_{h \in I} \pi_{-p}(h)$. For any set $B \subset \mathcal{I}_p$, let $\xi_p(B) = \max_{\sigma^*_{\mathrm{opp}(p)}} \sum_{I \in B} \chi^{\sigma^*}_{-p}(I)$ give the chance probability of reaching any information set in B if the opponent plays to reach information sets in B. Let Σ^D_p be the set of all pure player p strategies. Given $\sigma^D_p \in \Sigma^D_p$, let $N_p(\sigma^D_p) = \{I \in \mathcal{I}_p | \pi^{\sigma^D}_p(I) = 1\}$. It is convenient to talk about the depth of an information set I, defined as $|\{k \in H | p(k) = p, k \sqsubset h\}|$ for any $h \in I$.

Let $d^{\max}_p$ be the maximum depth of any player p information set. For any pure strategy $\sigma^D_p \in \Sigma^D_p$, let $\mathcal{B}_p(\sigma^D_p) = \{B^0_p, ..., B^{d^{\max}_p}_p\}$ be a partition of $N_p(\sigma^D_p)$, grouping information sets by depth. Let $\{\mathcal{B}^0_p(\sigma^D_p), \mathcal{B}^1_p(\sigma^D_p), ...\}$ be a collection of sub-partitions of each $B^d_p \in \mathcal{B}_p(\sigma^D_p)$.

Theorem 1 *Let L be a bound on values such that $|u_p(h) - u_p(j)| \leq L$ for all h, j, p, and $A_p = \max_{I \in \mathcal{I}_p} |A(I)|$ be the maximum number of player p actions. Say we are given partitions $\{\mathcal{B}^0_p(\sigma^D_p), \mathcal{B}^1_p(\sigma^D_p), ...\}$ for any pure strategy $\sigma^D_p \in \Sigma^D_p$. Then after T CFR iterations, $R^T_p \leq L M_p \sqrt{A_p T}$, where $M_p = \max_{\sigma^D_p \in \Sigma^D_p} \sum_{d=0}^{d^{\max}_p} \sum_{B \in \mathcal{B}^d_p(\sigma^D_p)} \xi_p(B) \sqrt{|B|}$.*

Proof.

$$\begin{aligned}
R^T_p &= \sum_{I \in \mathcal{I}_p} \pi^{\sigma^*}_p(I) R^T(I) && \text{Theorem 4 of [30]} \\
&= \max_{\sigma^D_p \in \Sigma^D_p} \sum_{I \in \mathcal{I}_p} \pi^{\sigma^D}_p(I) R^T(I) \\
&= \max_{\sigma^D_p \in \Sigma^D_p} \sum_{I \in N(\sigma^D_p)} R^T(I) \\
&= \max_{\sigma^D_p \in \Sigma^D_p} \sum_{d=0}^{d^{\max}_p} \sum_{I \in B^d_p} R^T(I) && \text{using } \mathcal{B}_p(\sigma^D_p)
\end{aligned}$$

Up to this point, we have written things differently, but could still follow the form of the proof for Theorem 5 in [30] to to get a bound similar to that of Gibson *et al.* We can continue to split this sum using the sub-partitions.

$$R^T_p = \max_{\sigma^D_p \in \Sigma^D_p} \sum_{d=0}^{d^{\max}_p} \sum_{B \in \mathcal{B}^d_p(\sigma^D_p)} \sum_{I \in B} R^T(I)$$

From regret-matching bounds, given an L^t such that $|v^t(I,a) - v^t(I,b)| \leq L^t$

$$\leq \max_{\sigma^D_p \in \Sigma^D_p} \sum_{d=0}^{d^{\max}_p} \sum_{B \in \mathcal{B}^d_p(\sigma^D_p)} \sum_{I \in B} \sqrt{|A(I)| \sum_{t=1}^{T} (L^t)^2}$$

$|v^t(I,a) - v^t(I,b)| \leq L\chi^{\sigma^t}_{-p}(I)$, so use $L^t = L\chi^{\sigma^t}_{-p}(I)$

$$\leq \max_{\sigma^D_p \in \Sigma^D_p} \sum_{d=0}^{d^{\max}_p} \sum_{B \in \mathcal{B}^d_p(\sigma^D_p)} \sum_{I \in B} \sqrt{|A(I)| \sum_{t=1}^{T} (L\chi^{\sigma^t}_{-p}(I))^2}$$

Multiply by $\xi_p(X)/\xi_p(B)$

$$= \max_{\sigma^D_p \in \Sigma^D_p} \sum_{d=0}^{d^{\max}_p} \sum_{B \in \mathcal{B}^d_p(\sigma^D_p)} \sum_{I \in B} \sqrt{|A(I)| \sum_{t=1}^{T} (L\chi^{\sigma^t}_{-p}(I)\xi_p(B)/\xi_p(B))^2}$$

Using $|A(I)| \leq A_p$ and pulling constants out of inner summations

$$\leq \max_{\sigma^D_p \in \Sigma^D_p} \sum_{d=0}^{d^{\max}_p} \sum_{B \in \mathcal{B}^d_p(\sigma^D_p)} L\xi_p(B)\sqrt{A_p} \sum_{I \in B} \sqrt{\sum_{t=1}^{T} (\chi^{\sigma^t}_{-p}(I)/\xi_p(B))^2}$$

By the definition of $\xi_p(B)$, for all $I \in B$, we have $0 \leq \chi^{\sigma^t}_{-p}(I)/\xi_p(B) \leq 1$. We also know that for all I, $\sum_{t=1}^{T} \sum_{I \in B} \chi^{\sigma^t}_{-p}(I)/\xi_p(B) \leq T$, so by Lemma 6 of [58], $\sum_{I \in B} \sqrt{\sum_{t=1}^{T} (\chi^{\sigma^t}_{-p}(I)/\xi_p(B))^2} \leq \sqrt{|B|T}$ and we have

$$R^T_p \leq L\sqrt{A_p T} \max_{\sigma^D_p \in \Sigma^D_p} \sum_{d=0}^{d^{\max}_p} \sum_{B \in \mathcal{B}^d_p(\sigma^D_p)} \xi_p(B)\sqrt{|B|} = LM_p\sqrt{A_p T}$$

■

3.2.2 Simplified Special Cases

Theorem 1 gives a relatively complicated bound, and does not specify how the depth d informations sets in B^d_p should be partitioned. To get the tightest possible bound from Theorem 1 we would use the partition which minimises M_p, but this is unlikely to be computable. In general, to get an improved bound over the $M(\sigma^*)$ bound of Gibson *et al.*, we would like to split up $B^d_p \in \mathcal{B}_p(\sigma^D_p)$ so that $|\mathcal{B}^d_p(\sigma^D_p)|$ is large and $\sum_{B \in \mathcal{B}^d_p(\sigma^D_p)} \xi_p(B)$ is small. There are at least two special cases that provide a simplified regret bound.

First, for any game, we can choose to not refine any part B^d_p.

Corollary 2 $R^T_p \leq L\sqrt{A_p T} \max_{\sigma^D \in \Sigma^D} \sum_{d=0}^{d^{\max}_p} \sqrt{|B^d_p|}$

Proof. If B_p^d is not refined, we have $\mathcal{B}_p^d(\sigma_p^D) = \{B_p^d\}$. Noting that $\xi_p(B_p^d) \leq 1$, Theorem 1 gives us the desired result. ■

The bound of Corollary 2 is at least as tight as the bound of Gibson *et al.* For any pure strategy σ_p^D, the information sets which have a non-zero coefficient in the bound of Gibson *et al.* are exactly those information sets which are in $N_p(\sigma_p^D)$. If $\mathcal{B}'_p$ is the partition of Gibson *et al.*, then $\mathcal{B}'_p$ is a refinement of the partition $\mathcal{B}_p(\sigma_p^D)$ when restricted to the information sets $N_p(\sigma_p^D)$. If two information sets $I, I' \in B' \in \mathcal{B}'_p$ have the same sequence of player p actions, they are at the same depth, and there exists d such that $I, I' \in B_p^d \in \mathcal{B}_p(\sigma_p^D)$. Because $\sqrt{|C|} \leq \sum_{C_i \in \mathcal{C}} \sqrt{|C_i|}$ for any partition $\mathcal{C}$ of C, the bound of Corollary 2 is no greater than the bound of Gibson *et al.*

Second, we can compute a tighter bound for the class of games where B_p^d can be partitioned so that the parts are reachable by disjoint sets of chance sequences. That is, $\sum_{B \in \mathcal{B}_p^d(\sigma_p^D)} \xi_p(B) = \xi_p(B_p^d)$. This occurs when there is some chance event(s) where the player always knows the value of some function of the chance event's outcome.

Corollary 3 *If* $\sum_{B \in \mathcal{B}_p^d(\sigma_p^D)} \xi_p(B) = \xi_p(B_p^d)$ *for each sub-partition* $\mathcal{B}_p(\sigma_p^D)$*, then* $R_p^T \leq L\sqrt{A_p T} \max_{\sigma_p^D \in \Sigma_p^D} \sum_{d=0}^{d_p^{\max}} \max_{B \in \mathcal{B}_p^d(\sigma_p^D)} \sqrt{|B|}$

Proof. By assumption $\sum_{B \in \mathcal{B}_p^d(\sigma_p^D)} \xi_p(B) = \xi_p(B_p^d)$. It is always the case that $\xi_p(B_p^d) \leq 1$ so we get $\sum_{B \in \mathcal{B}_p^d(\sigma_p^D)} \xi_p(B)\sqrt{|B|} \leq \max_{B \in \mathcal{B}_p^d(\sigma_p^D)} \sqrt{|B|}$. Applying this to Theorem 1 gives us the result. ■

3.2.3 Application to HULHE

We demonstrate the utility of the improved regret bound by considering the game of HULHE. Even though HULHE does not have disjoint chance events at all depths, Theorem 1 can still be used to get a bound at least 26 000 times smaller than the bound generated by the work of Gibson *et al.* [30].

We start by looking at the bound of Corollary 2 where we do not refine B_p^d, recalling that this simplification is at least as tight as the bound of Gibson *et al.* We must find the maximum over all pure strategies σ_p^D of $L\sqrt{A_p T} \sum_{d=0}^{d_p^{\max}} \sqrt{|B_p^d|}$, where B_p^d only contains information sets reachable by

σ_p^D. The maximisation can immediately be simplified by considering only the betting tree, because HULHE has a composite structure where the possible chance events and the betting tree are independent of each other. The only interaction is that the number of chance events in a game is determined by the number of betting rounds, so the maximising choice will be the same for all chance events.

In general, we cannot use the usual technique of recursively and independently maximising within subtrees, because the "value" of a subgame is a vector of sizes at different depths. Consider some game where the partition by depth produces three parts, and there are two top level information sets X and Y. From X, depending on the strategy chosen, there are either 1, 2, and 10 reachable information sets at depth 1, 2, 3 respectively, or 1, 3, and 3 information sets. There are always 1, 1, and k information sets reachable from Y. Depending on the strategy, this gives us a bound of $L\sqrt{A_pT}(\sqrt{1+1}+\sqrt{2+1}+\sqrt{10+k})$ or $L\sqrt{A_pT}(\sqrt{1+1}+\sqrt{3+1}+\sqrt{3+k})$. The decision of which bound is larger, which determines the strategy at X, depends on the value of k, which is determined by the strategy at Y. If k is 164 or less, the first choice at X is greater, otherwise the second choice is greater. Fortunately this does not occur in HULHE, as there is always a choice in each subtree which dominates all other choices, so we can efficiently find the maximising strategy.

In HULHE, for both players $A_p = 3$ and the most they can win or lose is 24 big blind units, so $L = \max_{j,h \in Z} |u_p(j) - u_p(h)| = 48$. Reducing the chance branching factor using suit equivalence [4], there are $c_1 = 169$, $c_2 = 1\,286\,792$, $c_3 = 55\,190\,538$, and $c_4 = 2\,428\,287\,420$ information sets for each betting sequence in rounds 1 to 4 respectively [44] due to different chance events. Finding the pure strategy which maximises for the strategy gives us the bounds

$$\begin{aligned}R_1^T \leq 48\sqrt{3T}(&\sqrt{2c_1} + \sqrt{2c_1 + 2c_2}\\
&+\sqrt{4c_2 + 3c_3} + \sqrt{4c_2 + 6c_3 + 2c_4} + \sqrt{2c_2 + 10c_3 + 8c_4}\\
&+\sqrt{10c_3 + 18c_4} + \sqrt{6c_3 + 24c_4} + \sqrt{2c_3 + 26c_4}\\
&+\sqrt{18c_4} + \sqrt{8c_4} + \sqrt{2c_4}) \approx 112\,253\,847\sqrt{T}\end{aligned}$$

$$\begin{aligned}R_2^T \leq 48\sqrt{3T}(&\sqrt{3c_1} + \sqrt{3c_1 + 4c_2} + \sqrt{1c_1 + 8c_2 + 6c_3}\\
&+\sqrt{9c_2 + 18c_3 + 10c_4} + \sqrt{5c_2 + 30c_3 + 40c_4} + \sqrt{1c_2 + 30c_3 + 86c_4}\\
&+\sqrt{19c_2 + 118c_4} + \sqrt{7c_3 + 112c_4} + \sqrt{1c_3 + 74c_4}\\
&+\sqrt{33c_4} + \sqrt{9c_4} + \sqrt{1c_4}) \approx 241\,016\,303\sqrt{T}\end{aligned}$$

We now consider a tighter bound using Theorem 1 and splitting by chance events. Despite the convenient structure, HULHE is not an example of a game where we can apply the simple bound of Corollary 3. For example, when considering the first player to act, the depth 1 partition might contain information sets with the betting sequences "raise call" and "raise raise". The sequences ending in call are now in the second round of betting where another chance event has occurred, while the sequences ending in raise are still in the first round of betting. Using Theorem 1 it is not possible to cleanly separate the first round chance events and second round chance events.

One possible way to partition each $B_p^d \in \mathcal{B}_p(\sigma_p^D)$ is to split the information sets by both the betting round and the player's view of the chance events. That is, we get partitions $\mathcal{B}_p^d(\sigma_p^D)$ for rounds $r \in \{pre{-}flop, flop, turn, river\}$. While $\sum_{r=1}^{4} \sum_{B \in \mathcal{B}_{p,r}^d(\sigma_p^D)} \xi_p(B) \geq \xi(B_p^d)$ we do get $\sum_{B \in \mathcal{B}_{p,r}^d(\sigma_p^D)} \xi_p(B) = \xi(B_{p,r}^d)$. Finding the maximising σ_p^D and applying Theorem 1 then gives upper bounds

$$\begin{aligned}
R_1^T \le 48\sqrt{3T}((&\sqrt{2}+\sqrt{0}+\sqrt{0}+\sqrt{0})+(\sqrt{2}+\sqrt{2}+\sqrt{0}+\sqrt{0})\\
&+(\sqrt{0}+\sqrt{4}+\sqrt{2}+\sqrt{0})+(\sqrt{0}+\sqrt{4}+\sqrt{6}+\sqrt{2})\\
&+(\sqrt{0}+\sqrt{2}+\sqrt{10}+\sqrt{8})+(\sqrt{0}+\sqrt{0}+\sqrt{10}+\sqrt{18})\\
&+(\sqrt{0}+\sqrt{0}+\sqrt{6}+\sqrt{26})+(\sqrt{0}+\sqrt{0}+\sqrt{2}+\sqrt{26})\\
&+(\sqrt{0}+\sqrt{0}+\sqrt{0}+\sqrt{18})+(\sqrt{0}+\sqrt{0}+\sqrt{0}+\sqrt{8})\\
&+(\sqrt{0}+\sqrt{0}+\sqrt{0}+\sqrt{2}))\approx 4230\sqrt{T}\\
R_2^T \le 48\sqrt{3T}((&\sqrt{3}+\sqrt{0}+\sqrt{0}+\sqrt{0})+(\sqrt{3}+\sqrt{4}+\sqrt{0}+\sqrt{0})\\
&+(\sqrt{1}+\sqrt{8}+\sqrt{6}+\sqrt{0})+(\sqrt{0}+\sqrt{9}+\sqrt{18}+\sqrt{10})\\
&+(\sqrt{0}+\sqrt{5}+\sqrt{30}+\sqrt{40})+(\sqrt{0}+\sqrt{1}+\sqrt{30}+\sqrt{86})\\
&+(\sqrt{0}+\sqrt{0}+\sqrt{19}+\sqrt{118})+(\sqrt{0}+\sqrt{0}+\sqrt{7}+\sqrt{112})\\
&+(\sqrt{0}+\sqrt{0}+\sqrt{1}+\sqrt{74})+(\sqrt{0}+\sqrt{0}+\sqrt{0}+\sqrt{33})\\
&+(\sqrt{0}+\sqrt{0}+\sqrt{0}+\sqrt{9})+(\sqrt{0}+\sqrt{0}+\sqrt{0}+\sqrt{1}))\approx 8292\sqrt{T}
\end{aligned}$$

The advantage of partitioning by chance events is evident in the large difference between the two pairs of bounds. Even though the chance partitioning bound is increased by the overlapping chance events, compared to the previous best bound of Gibson *et al.* the new bounds are at least 26 538 and 29 067 times smaller for player 1 and 2 respectively.

It is important to note that the bounds of Theorem 1 and previous work are mostly informative about the asymptotic behaviour as the problem grows. The new bounds represent a significant improvement, but they may still be very loose. For example, in HULHE the absolute worst case we can add at most 48 big blind units of regret per time step, even if we were not minimising regret. We can therefore always say that $R_p^T \le 48T$. It would require 7767 or more iterations for the $R_1^T \le 4230\sqrt{T}$ bound of Theorem 1 to be smaller than the trivial non-regret-minimising bound.

3.3 Complexity Analysis

Theorem 1 gives us a regret bound which we can use to compute a bound on the running time of CFR. Surprisingly, this analysis of running time has not appeared in any of the CFR papers, which only give the regret bound. As before, let L be a bound on the difference in values so that $|u(h) - u(j)| \leq L$ for all h, j, and let $A = \max_I |A(I)|$ be a player-independent bound on the number of actions at any information set.

Theorem 4 *CFR is guaranteed to produce an ϵ-Nash equilibrium with a running time of $\mathcal{O}(|H|AL^2(\sum_p M_p)^2/\epsilon^2)$, where M_p is defined as in Theorem 1.*

Proof. Corollary 2 states that after T CFR iterations, a player p will have at most $LM_p\sqrt{AT}$ regret. From the connection between regret and exploitability, the profile of average strategies $\langle\bar{\sigma_1}, \bar{\sigma_2}\rangle$ is an $L\sqrt{A/T}\sum_p M_p$-Nash equilibrium. Re-arranging to find T for a fixed ϵ, we see that CFR might require $AL^2\sum_p(M_p)^2/\epsilon^2$ iterations to find an ϵ-Nash equilibrium.

At each iteration, CFR will have to do $\mathcal{O}(|H|)$ work to compute counterfactual values and update regrets. Combining this with the number of iterations gives a running time of $\mathcal{O}(|H|AL^2(\sum_p M_p)^2/\epsilon^2)$. ■

Theorem 5 *CFR requires $2\sum_p(|I_p|A_p) + \sum_p(|I_p|A_p)$ space.*

Proof. Storing the regrets and average strategy require us to store one value for each action at every information set, requiring a total of $2\sum_p |I_p|A_p$ entries. Algorithm 2 also uses temporary storage v to store the counterfactual values for an iteration, for another $\sum_p |I_p|A_p$ entries. ■

Both Theorem 4 and Theorem 5 apply to CFR in any extensive form game. In games where Algorithm 3 applies, we can eliminate the temporary storage requirements, at the cost of passing around vectors instead of scalars. For example, in HULHE the vectors are on the order of thousands of entries (the number of possible hands), which is extremely small compared to the size of $\sum_p |I_p|A_p$, leading to an impression that CFR only requires 2 entries per action at every information set, which is mistaken in the general case.

3.3.1 Stopping Conditions

All of the descriptions of CFR list the number of iterations T as a value provided by the user. In general, however, we are more likely to choose the value of T at run-time, as a side effect of some other desired stopping condition[2]. One possibility is that we might have a limited amount of time, and will simply run iterations until we run out of time. For example, when playing HUNL against human professionals, described in Chapter 6, we had a limited computation time, and selected a number of iterations to fit within those limits.

Another possibility is that we have a maximum exploitability we will accept in an approximate solution, and want to run just enough iterations to reach that target. For example, if the standard deviation of observed outcomes is less than 0.5 and we are using the resulting strategy 10 000 times, we might be satisfied with a 0.01-Nash equilibrium, as the exploitability is roughly as large as the 95% confidence interval on the average observed result. We can periodically measure the exploitability, and stop running iterations if the exploitability is low enough. Computing the best response value for each player to calculate the exploitability is on the same order of effort as a CFR iteration, so if we check infrequently we do not greatly increase the total running time. We used this stopping condition when solving HULHE, described in Chapter 4.

3.3.2 Comparison to Other Algorithms

There are two main classes of alternatives to CFR variants when solving extensive form games: solving a linear program (LP), and fast techniques for convex optimisation problems.

Solving an extensive form game as a matrix game [88] using mixed strategies requires space and time that is exponential in the size $|H|$ of the game. Using an LP became practical after Koller *et al.* introduced the idea of a sequence form LP which could be solved to find behaviour strategies [51]. This requires a sparse matrix of size roughly $|I_1|A_1+|I_2|$ by $|I_2|A_2+|I_1|$, with $\mathcal{O}(|Z|)$ non-zero entries. Using Karmarkar's algorithm for solving an LP [49], which

[2]Changing T at run-time is reasonable for CFR and its variants, as there is no dependence within the computation on T, other than the decision to run further iterations.

has a low worst-case complexity, would require $\mathcal{O}((|I_p|A_p + |I_{2-p}|)^{3.5} \log_2(1/\epsilon))$ operations, choosing the primal player $p \in P$ to minimise this work.

We can also consider convex optimisation within the framework where we are restricted to local queries about the gradient at the current point. Nesterov *et al.* showed that accelerated techniques requiring smooth functions could be applied to a non-smooth convex function to get a $\mathcal{O}(1/\epsilon)$ rate of convergence [71] to the optimum, and specifically could be applied to saddle-point problems like finding a Nash equilibrium for matrix games with the Excessive Gap Technique (EGT) [70].

Gilpin *et al.* extended EGT to the sequence form linear program of Koller *et al.*, providing the necessary strongly convex distance function between two behaviour strategies [31]. EGT has the same $\mathcal{O}(|H|)$ cost per iteration as CFR, for a running time of $\mathcal{O}(|H| \log(A) L \sum_{p \in P} |I_p| d_p^{\max} 2^{d_p^{\max}} M'_p/\epsilon)$ to guarantee an ϵ-Nash equilibrium [54]. The M'_p constant plays a similar role to the M_p constant in Corollary 2, except that M'_p "essentially measures the branching factor over the other player and nature" [54]. Applying EGT to an extensive form game requires storing $3 \sum_p (|I_p|A_p)$ entries, plus temporary storage of $\sum_p (|I_p|A_p)$ entries, which is asymptotically identical to CFR.

The distance function for behaviour strategies can be used with other accelerated convex optimisation algorithms, like Nemirovski's prox-method [68], or any variants of Nesterov's accelerated convex optimisation algorithm [71]. All of these accelerated optimisation-based techniques have the same asymptotic worst-case behaviour as EGT, with the same space requirements.

CFR uses regret-matching at each information set to guarantee low regret. There is, however, no reason why other online regret minimisation algorithms could not be used. For example, CFR could use the Hedge algorithm [26]. The operations for Hedge are more computationally expensive than regret-matching, but Hedge also has an asymptotically better regret bound with respect to the number of actions A: proportional to $\mathcal{O}(\log(A))$ rather than $\mathcal{O}(\sqrt{A})$.

While the different algorithm bounds are difficult to compare directly, there is a general sense that CFR scales better to larger games, while other choices of algorithm perform better when high accuracy is needed. The theoretical difference in accuracy over time is easy to show: the $\mathcal{O}(1/\sqrt{\epsilon})$ of CFR is worse than the $\mathcal{O}(1/\epsilon)$ of EGT or the $\mathcal{O}(\log_2(1/\epsilon))$ of Karmarkar's algorithm.

The scaling with game size is harder to show. To get a rough sense, we can consider a game with unit payoffs where for any player p and depth d there are $(i_p j_p)^d$ information sets, and j_p^d information sets reachable by a pure player p strategy. This gives us $|I_p| = \frac{(i_p j_p)^{d_p^{\max}+1}-1}{i_p j_p - 1}$ and $M_p = \frac{\sqrt{j_p}^{d_p^{\max}+1}-1}{\sqrt{j_p}-1}$ using Corollary 2. If $i_p \geq 2$, $j_p \geq 2$, and $d_p^{\max} \geq 3$ then $M_p^2 \leq |I_p|$. We also have $|H| \leq |I_p|^2$, so for fixed ϵ Theorem 4 gives us $\mathcal{O}(A|I_p|^3)$ which is better than the $\mathcal{O}((A|I_p|)^{3.5})$ of Karmarkar's algorithm or the $\mathcal{O}(\log(A)|I_p|^3 d_p^{\max} 2^{d_p^{\max}} M'_p)$ from the current best EGT bounds.

As a specific example, we can consider HULHE. From Section 3.2.3 we get $L\sqrt{A}M_1 = 4230$ and $L\sqrt{A}M_2 = 8292$. $|H| = 3.162 * 10^{17}$ [58] so CFR has a run time bound of $3.162 * 10^{17} AL^2(\sum_p(M_p))^2/\epsilon^2 \approx 4.96 * 10^{25}/\epsilon^2$. Looking at Karmarkar's algorithm $|I_1|A + |I_2| \approx 2.48 * 10^{13}$, giving it a run time bound of $7.6 * 10^{46}/\log_2(1/\epsilon)$. Finally looking at EGT, $|I_1| = |I_2| \approx 6.90 * 10^{12}$, $d_{1,\max} = 11$, and $d_{2,\max} = 12$ giving it a run time bound of $(3.74 * 10^{36} M'_1 + 8.16 * 10^{36} M'_2)/\epsilon$ for some large constants M'_1 and M'_2. It must be noted again that these bound are almost certainly loose upper bounds (especially for EGT), so that the numbers do not necessarily correspond to actual performance, and might still be improved with better theoretical work in the future[3].

3.4 Monte Carlo CFR

Monte Carlo CFR (MCCFR) refers to a family of CFR variants that update a randomly sampled subset of information sets at each information set [59]. The general family of MCCFR uses a partition $\mathcal{Q}$ of terminal states Z. At each iteration, it samples one block $Q_i \in \mathcal{Q}$ and updates all information set based on that block. For an intelligent choice of partition $\mathcal{Q}$, each block will provably have no effect on most information sets, making for very fast iterations. The tradeoff is less progress per iteration.

[3]For example, using the dilated entropy prox function of Kroer *et al.*[55].

3.4.1 MCCFR Regret Bound

The existing analysis of MCCFR [30] can be directly updated using the new CFR regret bound of Theorem 1. Given a partition $\mathcal{Q} = \{Q_1, ...\}$ where Q_i is sampled with probability q_i, let $\delta = \min_{z \in Z} \sum_{i:z \in Q_i} q_i$. With probability $1 - \rho$

$$R_p^T \leq \frac{L}{\delta} \left(M_p + \frac{\sqrt{2|I_p||\cup_d \mathcal{B}_p^d|}}{\rho} \right) \sqrt{A_p T}$$

As with the updated CFR bound, the only change in the updated MCCFR bound is in the M_p constant describing the scaling with respect to game size. The $\mathcal{O}(\sqrt{T}/\rho)$ bound means that for any $\epsilon > 0$ and certainty $1 - \rho$ we can probably find an ϵ-Nash equilibrium if we run enough iterations. The factor of $1/\delta$ describes how the required number of iterations increases as the number of outcomes sampled per iteration decreases.

3.5 Why is CFR Popular?

Following its introduction in 2007, CFR variants have become a popular choice of algorithm for solving large extensive form games. Given that other algorithms have better asymptotic behaviour with respect to solution quality, why is CFR popular, and a state-of-the-art algorithm?

One of the most important factors in CFR's success is that it performs well in practice, taking advantage of the potential disconnect between asymptotic behaviour and real-world performance for a specific problem. Asymptotic worst-case bounds can hide two important properties. First, asymptotic bounds often ignore multiplicative constants, and the constant factor for CFR is small. CFR has fast iterations, few passes through the game per iteration, and it uses simple addition and division operations rather than exponentiation.

Second, asymptotic worst-case bounds do not say anything about the performance in all problem instances, only about the worst possible problem of a given size. On many problems, CFR improves exploitability more quickly than the $1/\sqrt{T}$ worst-case bounds; in some cases, the experimentally observed exploitability improvement over time for CFR is closer to the $1/T$ expected from EGT, as seen in Section 4.3.2.

As the available running time or required accuracy increases, we would expect to CFR eventually become less competitive than other asymptotically faster algorithms, but that crossover point might be beyond the target time or accuracy. CFR is a good choice for parts-per-thousand accuracy, and many situations do not require a more precise equilibrium approximation.

One example of a situation where only moderate accuracy is needed arises when we have limited play time. Chance events and mixed strategies add variance to the expected utility of a strategy profile, so that the exploitability might be well under what is statistically detectable during play. This variance-based notion of "good enough" was formalised in our notion of a game being essentially solved [9].

Low space requirements contribute to CFR's popularity. For some problems, like solving games, it is acceptable to spend a long time for an offline computation, and the limiting factor which determines whether the game can be solved becomes the total space required. CFR requires less space than other commonly used algorithms: 2 values per player sequence for CFR, 3 values per player sequence for EGT, and roughly the product of the number of player sequences for LPs.

As a final factor, CFR is easy to understand, modify, and implement, even in a distributed computing environment. The flexible framework of regret minimisation allows special purpose variants of CFR like CFR-BR [45] which uses a best response for one player, or MCCFR using game specific sampling schemes. Regret-matching theoretically scales more poorly than Hedge with large numbers of actions, but the performance of Hedge is very dependent on the step size parameter which would need to be individually tuned at every information set. The projection step in EGT requires a similarly tuned step size parameter. CFR is a parameter-free algorithm, which works well without requiring any parameter tuning.

Chapter 4

Using Less Time: CFR+

When solving a game offline, the computation time required is a soft limit because for small factors k, we can always run for k times longer. Despite this soft limit for computation compared to the hard limitation of available space, time requirements increase faster than space requirements (Section 3.3) so that CFR can still be runtime-limited on large machines. As hardware improved and more space became available, it became possible to run game-solving algorithms on games that were so large that the computation time was unreasonable. To address the running time, Tammelin introduced CFR$^+$ [84], a new CFR variant that had the same space requirements as CFR, but empirically faster convergence.

My contributions are worst-case bounds for CFR$^+$ and two different worst-case bounds for its component algorithm regret-matching$^+$ (published at IJCAI [85]), a comparison of CFR$^+$ to game solving algorithms that have theoretically faster convergence rates, and the majority of a high performance distributed implementation of CFR$^+$ designed for the landmark result of solving the game of HULHE (published in Science [9]). Together, the worst case bounds and empirical results provide a compelling reason for people to consider using regret-matching$^+$ over regret-matching, and CFR$^+$ over non-sampling variants of CFR.

4.1 CFR$^+$ Algorithm

This section gives a description of Tammelin's CFR$^+$ algorithm, and the regret-matching$^+$ algorithm it is based on.

CFR combines a regret minimising algorithm with an averaging step to produce an ϵ-Nash equilibrium. CFR$^+$ modifies both of these parts. CFR uses the regret-matching algorithm, and returns a uniformly weighted average strategy as the game solution. CFR$^+$ uses a new regret minimising algorithm, regret-matching$^+$, and a non-uniformly weighted average strategy. Finally, CFR$^+$ specifies that players must be sequentially updated in alternation, rather than the simultaneous update used by the original CFR paper. That is, we update the regrets for player 1, then update the regrets for player 2 relative to the new strategy for player 1, and so on.

4.1.1 Regret-matching$^+$ Algorithm

Regret-matching$^+$, a regret minimising algorithm, is an online algorithm for action selection in a full information, adversarial setting. It operates very similarly to the regret-matching algorithm and has a similar theoretical bound on external regret. The difference between the two algorithms is that regret-matching$^+$ sets any negative regret values to zero at each iteration, while the original regret-matching algorithm stores negative regret values but treats them as zero when computing the current strategy. In short, an implementation of regret-matching can be updated to regret-matching$^+$ by setting any stored negative values to zero, and skipping the regret-matching sign check when computing the strategy.

In a formal description of the setting there is a fixed set of actions A, and at each time step t from 1 to time T, we commit to a probability distribution $\sigma^t()$ for selecting actions. An adversary knows $\sigma^t()$ but does not know the result of the private random source we will use to select an action from $\sigma^t()$. Each action $a \in A$ has a value $v^t(a)$, all of which are unknown to the agent until after time t.

As described in Section 2.7.2, an implementation of regret-matching stores the values $R^t(a) = R^{t-1}(a) + \Delta R^t(a)$ for each action a, where $R^0(a) = 0$ and $\Delta R^t(a) = v^t(a) - \sum_{b \in A} \sigma^t(b)v^t(b)$. The strategy is the normalised positive action regrets: $\sigma^t(a) = R^{t-1}(a)^+ / \sum_{b \in A} R^{t-1}(b)^+$.

In regret-matching$^+$, instead of using $R^t(a)^+ = \max(R^t(a), 0)$ when computing σ^{t+1}, we store it. Regret-matching$^+$ stores the regret-like values $Q^t(a) = (Q^{t-1}(a) + \Delta R^t(a))^+$ instead of the action regrets $R^t(a)$ stored by regret-matching. Note that regret-matching$^+$ uses the same $\Delta R^t(a)$ value as regret-matching. The value $\Delta Q^t(a) = Q^t(a) - Q^{t-1}(a)$ is useful in the accompanying theory, but it is not computed in a regret-matching$^+$ implementation.

The regret-matching$^+$ strategy is the normalised regret-like values: $\sigma^t(a) = Q^{t-1}(a) / \sum_{b \in A} Q^{t-1}(b)$. In a regret-matching$^+$ implementation this looks the same as computing $\sigma^t()$ for regret-matching, except regret-matching$^+$ no longer needs to check the sign of the stored values to treat negative numbers as 0.

Note that because the handling of negative values is the only difference between regret-matching and regret-matching$^+$, the two algorithms are almost identical in terms of work per time step. Regret-matching$^+$ checks the sign of each stored value when updating the stored values, where regret-matching checks the sign of each stored value when computing the strategy. In situations where the strategy is computed multiple times before updating the regret values, regret-matching$^+$ is more efficient. There is no corresponding situation where regret-matching is more efficient because the strategy must be computed at least once in order to update the regret values.

4.1.2 CFR$^+$ Average Strategy

CFR$^+$ uses a different weighting scheme than CFR to produce an average strategy that is returned as an ϵ-Nash equilibrium. Where the original CFR algorithm uses uniform weighting across each time step, CFR$^+$ instead uses linearly increasing weight so that σ^t from step t is given weight t in the average, which places more weight on recent strategies.

For each player p, the CFR$^+$ algorithm returns an average strategy $\bar{\sigma}_p = \frac{2}{T(T+1)} \sum_{t=1}^{T} t\sigma_p^t$. As with CFR, CFR$^+$ is working with behaviour strategies but must do the averaging within the space of sequence form strategies. The correct average can be maintained by storing and incrementally updating $acc(I, a) = \sum_{t=1}^{T} t\pi_p^t(I)\sigma^t(I, a)$ for each information set and action. Using the $acc(I, a)$ values at information set I we can compute action probability for action a in the average strategy as $\bar{\sigma}_p(I, a) = acc(I, a)/\sum_{a' \in A(I)} acc(I, a')$.

4.1.3 CFR$^+$ Alternating Update

The third change in CFR$^+$ is the order in which regrets are updated. As originally described by Zinkevich *et al.*, the CFR algorithm simultaneously updates the regrets for both players. That is, at time t for all information sets I, $R^t(I)$ is updated using σ_1^t and σ_2^t.

CFR$^+$ uses an alternating update. At time t, we first update all player 1 regrets. For all player 1 information sets $I \in \mathcal{I}_1$, $Q^t(I)$ is updated using σ_1^t and σ_2^t. Using the updated player 1 regrets, we get a new strategy $\sigma_1'^t$. Next, player 2 information sets are updated using the new player 1 strategy. For all player 2 information sets $I \in \mathcal{I}_2$, $Q^t(I)$ is updated using $\sigma_1'^t$ and σ_2^t.

In practice, some CFR implementations have also used alternating updates rather than simultaneous updates. To handle large games, an implementation of CFR or CFR$^+$ must be efficient in time and space. To save space, regrets must be used to compute values of $\sigma^t(I, a)$ as they are needed, rather than computing and storing a complete strategy. To save time, an efficient implementation should consider cutting off regret updates for any information sets the opponent has no probability of reaching, and player p average strategy updates for any information sets player p has no probability of reaching [43]. Both of these tricks are easier to implement when using alternating updates that modify values for a single player.

4.1.4 Algorithm

A simple, un-optimised implementation of CFR$^+$ is given in Algorithm 4. It assumes the the two players are named p_1 and p_2. The functions `UpdateAverage`, `UpdateAverage`, `StrategyToValues`, and `NormaliseAverage` are unchanged from the CFR description of Algorithm 1 in Section 3.1. Alternating updates are accomplished by running the four steps of an iteration on p_1 information sets, and then again on p_2 information sets. The linear weighted average uses a weight of t instead of 1 for the current strategy σ_t.

```
input  : Game $\mathcal{G}$, Number of iterations $T$
output: $\epsilon$-Nash approximation $\bar{\sigma}$
foreach $I \in \mathcal{I}$ do
    foreach $a \in A(I)$ do
        $R(I, a) \leftarrow 0$
        $\bar{\sigma}(I, a) \leftarrow 0$
$\sigma \leftarrow$ RegretToStrategy($\mathcal{I}$,$R$)
for $t \leftarrow 1$ to $T$ do
    foreach $p \in \{p_1, p_2\}$ do
        UpdateAverage($\mathcal{I}_p$,$\sigma$,$t$,$\bar{\sigma}$)
        $v_p \leftarrow$ StrategyToValues($\mathcal{I}_p$,$\sigma$)
        UpdateRegrets$^+$($\mathcal{I}_p$,$\sigma$,$v_p$,$R$)
        $\sigma_p \leftarrow$ RegretToStrategy($\mathcal{I}_p$,$R$)
NormaliseAverage($\mathcal{I}$,$\bar{\sigma}$)
return $\bar{\sigma}$

function UpdateRegrets$^+$($UpdateSets$,$\sigma$,$v$,$R$):
    foreach $I \in UpdateSets$ do
        foreach $a \in A(I)$ do
            $R(I, a) \leftarrow \max(0, R(I, a) + v(I, a) - \sum_{b \in A(I)} \sigma(I, b)v(I, b))$
```

Algorithm 4: CFR$^+$

4.1.5 Modified Value Computation

The connection to optimisation methods, discussed in Section 4.4, suggests a small change to the way CFR and CFR$^+$ compute the counterfactual values used in updating regrets. An implementation of CFR or CFR$^+$ which uses a depth first traversal, like Algorithm 2, returns a counterfactual value which is the dot product of the child values and the current strategy. We could instead update the regrets, get the new strategy, and return the dot product of the child values and the new strategy. Algorithm 5 shows this modification.

None of the theoretical analysis presented here applies to the modified CFR or CFR$^+$ algorithm, but the modified versions have slightly better empirical performance, as seen in Section 4.4.4. All other experimental results in this chapter use the modified algorithms.

```
function ModifiedRecursiveUpdate($h$,$s[\cdot]$):
    input  : State and sequence probabilities $\pi_p(h)$ for all $p$
    output: Counterfactual values for all $p$
    if $h \in Z$ then
        return $[s[p_2]\pi_c(h)u_{p_1}(h), s[p_1]\pi_c(h)u_{p_2}(h)]$
    $I \leftarrow I \in \mathcal{I}$ such that $h \in I$
    $r \leftarrow [0, 0]$
    if $p(h) = c$ then
        foreach $a \in A(h)$ do
            $r \leftarrow r +$ ModifiedRecursiveUpdate($h \cdot a$, $s$)
    else
        $s' \leftarrow s$
        $opp \leftarrow p \in P$ such that $p \neq p(h)$
        $\sigma \leftarrow$ RegretMatchingStrategy($I$)
        foreach $a \in A(h)$ do
            $s'[p(h)] \leftarrow s[p(h)] * \sigma[a]$
            $r' \leftarrow$ ModifiedRecursiveUpdate($h \cdot a$, $s'$)
            $v(I, a) \leftarrow v(I, a) + r'[p(h)]$
            $temp[a] \leftarrow r'[p(h)]$
            $r[opp] \leftarrow r[opp] + r'[opp]$
        $\sigma \leftarrow$ RegretMatchingStrategy($I$)
        foreach $a \in A(h)$ do
            $r[p(h)] \leftarrow r[p(h)] + temp[a]\sigma(a)$
        return $r$
```
Algorithm 5: Modified Value Computation

4.2 Theoretical Analysis

This section presents my contributions to the theoretical analysis of the CFR$^+$ algorithm and its components. I give an external regret bound for regret-matching$^+$ and an upper bound on exploitability for CFR$^+$. Both bounds show the worst case asymptotic behaviour of the variant algorithms to be the same as the original regret-matching and CFR algorithms. These results were originally published at IJCAI [85]. I also give an unpublished improved tracking regret bound for regret-matching$^+$, updating the published bound [85].

Proofs of the bounds presented in this section are given in Appendix A. All work in this section assumes there is a bounded difference between values. Specifically, there is some $L \in \mathbb{R}$ such that for all times t and choices a and b, $|v^t(a) - v^t(b)| \leq L$.

4.2.1 External Regret Bound for Regret-matching$^+$

The first result is that after T steps, regret-matching$^+$ has the same $L\sqrt{|A|T}$ upper bound on external regret as regret-matching. As discussed in Section 2.7, external regret is the most common regret measurement, based on a hindsight value comparison against a single static choice: how well would we have done with action a, compared to the expected value of the policies we did choose.

Theorem 6 *Given a set of actions A, any sequence of T value functions $v^t : A \mapsto \mathbb{R}$, and bound L such that $|v^t(a) - v^t(b)| \leq L$ for all t and $a, b \in A$, after playing the sequence σ^t of regret-matching$^+$ strategies, the regret-like value $Q^T(a) \leq L\sqrt{|A|T}$ for all $a \in A$. Moreover, this same bound applies to $R^T(a)$.*

4.2.2 Tracking Regret Bound for Regret-matching$^+$

The next result considers a different regret measurement that looks at non-static strategies. Tracking regret [37] is based on a hindsight value comparison against a piecewise static choice: how well would we have done with k static segments of different actions a_1, ..., a_k, compared to the expected value of the policies we did choose. Tracking regret is one possible generalisation of external regret, as external regret is tracking regret with $k = 1$.

Given an interval $B = [B_{\mathrm{s}}, B_{\mathrm{e}}]$ which is a subset of the T time steps $[1, T]$, we can define the regret $R^{[B]}(a)$ for action a in the interval B to be

$$R^{[B]}(a) = \sum_{t=B_{\mathrm{s}}}^{B_{\mathrm{e}}} \Delta R^t(a) \tag{4.1}$$

We can then write the $k-$tracking regret at time T as

$$R^{k,T} = \max_{\mathcal{B} \in \mathbb{B}_T} \sum_{B \in \mathcal{B}} \max_{a_B} R^{[B]}(a_B) \tag{4.2}$$

where $\mathbb{B}_T = \{\mathcal{B} | \bigcup_{B \in \mathcal{B}} B = [1, T], |\mathcal{B}| \leq k\}$ is the set of all partitions $\mathcal{B}$ of the T time steps into at most k intervals.

Theorem 7 *Given a set of actions A, any sequence of T value functions $v^t : A \mapsto \mathbb{R}$, and bound L such that $|v^t(a) - v^t(b)| \leq L$ for all t and $a, b \in A$, after playing the sequence σ^t of regret-matching$^+$ strategies, there is a bound on k-tracking regret of $R^{k,T} \leq kL\sqrt{|A|T}$.*

There are other regret minimising algorithms which have a better $\mathcal{O}(\sqrt{kT})$ bound on k-tracking regret, like AdaNormalHedge [62], but the $k\mathcal{O}(\sqrt{T})$ bound is still a non-trivial property that is not guaranteed by standard algorithms which minimise external regret. For example, the well known Hedge and regret-matching algorithms both have $\mathcal{O}(T)$ tracking regret bounds, even for the simplest case where $k = 2$, which can lead to unbounded average regret.

The linear 2-tracking regret of regret-matching can be demonstrated with a very simple problem. Consider the situation where $A = \{l, r\}$, $T = 2N$, $v_l^t = 1$ and $v_r^t = -1$ for the first N time steps, and $v_l^t = -1$ and $v_r^t = 1$ for the last N time steps: action l is the best choice, then action r.

After the first time step where the strategy is $P(l) = P(r) = 0.5$, regret-matching will always play action l, achieving a cumulative value of -1. For action l the external regret $R_l^t = 1$ for all t. For action r the external regret $R_r^t = 1 - 2N + 2|N - t|$ will decrease by 2 for N time steps and then increase for N time steps. The maximum external regret is a constant, the best possible result. However, considering the 2-tracking regret we see $2N+1 = T+1$ regret for the two part strategy which plays l for the first N steps and r for the next N steps. This piecewise strategy would have achieved a value of $2N$, compared to the cumulative value of -1 of the regret-matching policy, which means it has a lower bound on the 2-tracking regret of $T+1$.

Hedge has $\mathcal{O}(T)$ 2-tracking regret in the same problem. As described in Section 2.7.3, the strategy at time t is $\sigma^t(a) = e^{\gamma V^{t-1}(a)} / \sum_{a'} e^{\gamma V^{t-1}(a')}$, where $V^t(a) = \sum_{i=1}^{t} v_a^t(a)$. In this example, we have $V^t(l) = N - |t-N|$ and $V^t(r) = |t-N| - N$, so after simplifying we get $\sigma^t(l) = 1/(1+e^{2\gamma(|t-1-N|-N)})$ and $\sigma^t(r) = 1/(1+e^{2\gamma(N-|t-1-N|)})$. Looking at the cumulative value $\sum_{t=1}^{T} \sigma^t \cdot v^t$, canceling terms, and noting that $\sigma^1 \cdot v^1 = 0$, we get the value $\sigma^{N+1}(r) - \sigma^{N+1}(l) = \frac{e^{-2\sigma N} - e^{2\sigma N}}{2+e^{-2\sigma N}+e^{2\sigma N}}$. This quickly approaches -1 as N increases, giving the Hedge policy a lower bound on the 2-tracking regret that is slightly less than $T+1$.

4.2.3 Exploitability Bound for CFR$^+$

The final result is a bound on the exploitability of the weighted average strategy returned by the CFR$^+$ algorithm.

Zinkevich's original proof that CFR generates an ϵ-Nash equilibrium had three steps [94]: note that Blackwell proved regret-matching drives average counterfactual regret towards zero [5] at each information set, show that the regret is a function of the counterfactual regrets (updated by Theorem 1 in this work), and exploit a well-known link between regret and exploitability. The proof that CFR$^+$ generates an ϵ-Nash equilibrium uses similar steps.

Theorem 6 gives us the first step. For the second step, we can reconstruct a variant of Theorem 1 for the case where we have weighted the values at each iteration in the same way as the average strategy. Using the standard link between regret and exploitability gives us a bound on the weighted average strategy. Partitions $\mathcal{B}_p^d(\sigma_p^D)$ are defined as in Section 3.2.1.

Theorem 8 *Let L be a bound on values such that $|u_p(h) - u_p(j)| \leq L$ for all h, j, p, and $A_p = \max_{I \in \mathcal{I}_p} |A(I)|$ be the maximum number of player p actions.*

Say we are given partitions $\{\mathcal{B}_p^0(\sigma_p^D), \mathcal{B}_p^1(\sigma_p^D), ...\}$ for any pure strategy $\sigma_p^D \in \Sigma_p^D$. Then after T CFR$^+$ iterations, the linearly weighted average strategy profile $\bar{\sigma}_p^T = 2/(T^2+T)\sum_{t=1}^T t\sigma_p^t$ is a $2\sum_p LM_p\sqrt{A_p/T}$-Nash equilibrium, where $M_p = \max_{\sigma_p^D \in \Sigma_p^D} \sum_{d=0}^{d_{\max}} \sum_{B \in \mathcal{B}_p^d(\sigma_p^D)} \xi_p(B)\sqrt{|B|}$.

Note that this is the same bound as for CFR, with an additional factor of 2 that comes from the proof linking regret and the weighted average strategy. While the regret bound of Theorem 8 offers no incentive for using the CFR$^+$ linear weighted average, the experimentally observed behaviour suggests the weighted average is a better choice, as seen in the example of Section 4.3.5.

4.2.4 Complexity Analysis

The space requirements of CFR$^+$ are identical to CFR, and the upper bound on running time is off by a constant factor of two.

Let L be a bound on the difference in values so that $|u(h) - u(j)| \leq L$ for all h, j, and let $A = \max_I |A(I)|$ be a player-independent bound on the number of actions at any information set. Using Theorem 8 to stand in for Theorem 1, we can use the proof of Theorem 4 to get

Theorem 9 *CFR$^+$ is guaranteed to produce an ϵ-Nash equilibrium with a running time of $\mathcal{O}(|H|AL^2(\sum_p M_p)^2/\epsilon^2)$.*

Because the underlying regret minimisation problems have identical space requirements to CFR and the computation is so similar, we can directly use the proof of Theorem 5 to get

Theorem 10 *CFR$^+$ requires $2\sum_p(|I_p|A_p + \max_p |I_p|)$ space.*

4.3 Experimental Analysis

The theoretical results give an upper bound on the worst-case running time that is worse than CFR by a factor of 4. In practice CFR$^+$ greatly outperforms CFR on a range of games. The initial application of CFR$^+$ was essentially solving HULHE, using only 1579 iterations to produce a strategy, Cepheus, that was exploitable for less than 1 mbb/g [9]. Surprisingly, Cepheus was the current strategy described by the final regrets, rather than the average strategy prescribed by Theorem 8.

To show that CFR$^+$ has broader applicability than just HULHE, I compare the performance of CFR and CFR$^+$ on a wider range of games in Section 4.3.2. Further exploring Cepheus' use of the current strategy, results are included for both the current and average strategies.

4.3.1 CFR and CFR$^+$ Experimental Setup

With the exception of the specifically named simultaneous update results in Figure 4.13, all CFR results use alternating updates like those specified by the CFR$^+$ algorithm.

Figure 4.1 describes the performance of CFR$^+$ in HULHE as used in the generation of Cepheus. The strategy was generated using publicly released code [8], set up so that the average strategy discarded the current strategy for the first 200 iterations. The values reported here for the average strategy are off by a small amount (around 10% at iteration 1456), due to a bug in reporting the exploitability values of the average strategy, discovered while validating results with an independent codebase. Unfortunately, the computation is so large that re-running to correct the values is not feasible, nor was it feasible to produce comparable results for CFR.

In addition to HULHE, I use Rhode Island hold'em [79], Leduc hold'em [82], and Kuhn poker [56], three synthetic poker games created for artificial intelligence research. Results are given in Figure 4.2, Figure 4.3, and Figure 4.4 respectively. Rhode Island hold'em has 52 128 128 information sets, or approximately 4×10^6 information sets when taking advantage of poker-specific knowledge to eliminate strategically identical sets of cards. Leduc hold'em has 936 information sets, or 288 information sets using poker-specific knowledge about the cards. Kuhn's three-card poker variant has 12 information sets. Between the four poker games, we have games with very different sizes.

To avoid using nothing but variants of poker, I also use two sets of matrix games. The first set of matrix games are ten 1000×1000 random matrix games.

Each game was independently generated, and payoffs for each pairing of the 1000 pure strategies in a game were identically and independently sampled from a normal distribution with mean 0 and a standard deviation of 1. The exploitability reported in Figure 4.5 is the average value across the 10 games. While random matrix games generally result in uninteresting games for their size, they have still frequently been used in the literature [68, 71, 87].

The second set of matrix games are a collection of small simultaneous action games: matching pennies, rock-paper-scissors, and a Blotto game. The matching pennies game is one of the simplest non-trivial games, described by a 2×2 payoff matrix. Each player has a single, simultaneous choice of picking heads or tails. I used skewed payoffs where the row player wins 1 if both players chose heads, wins 4 if both players chose tails, and loses 2 otherwise. Many algorithms, including CFR, start with a uniform random strategy, and the skewed payoffs let us avoid the uninteresting situation where the algorithm starts at the exact equilibrium. Results for the matching pennies game are given in Figure 4.8.

Rock-paper-scissors is a well known game played using hand gestures. Each player picks rock, paper, or scissors and then simultaneously reveals their choice. Rock beats scissors, scissors beats paper, paper beats rock, and the same choice is a tie. As with matching pennies, I use a skewed version where rock versus paper has a magnitude of 2, rock versus scissors has a magnitude of 1, and paper versus scissors has a magnitude of 3. Results for rock-paper-scissors are given in Figure 4.7.

Blotto games are parameterised games where each player selects a sequence of non-decreasing integer values that has to add up to a specified value. The player's sequences are compared in pairwise fashion, and whichever player has a larger value in more pairs wins. The game is a draw if no player wins. I use the variant with sequences of length 3 that must sum to 16, which results in a 21×21 matrix game. Results for Blotto are given in Figure 4.6.

We can use log-log linear regression on the data to find the approximate rate of convergence. For example, if exploitability ϵ is decreasing as $1/\sqrt{T}$, then the best fit for $\log(\epsilon) = \beta * \log(T) + \alpha$ will have $\beta = -0.5$. Similarly,

$1/T$ performance will show up as -1.0. To reduce the effect of the swings in exploitability on the regression results, I only use the subset of points $\{(T, \epsilon) | \forall (T', \epsilon') \text{ s.t. } T' > T, \epsilon' < \epsilon\}$. This set gives us an (observed) upper bound on the exploitability at any future time.

The best fit β values are given in Table 4.1. Because the performance may change after the initial few iterations, I also report the convergence rates over the last 90% of the iterations in Table 4.2.

4.3.2 CFR and CFR$^+$ Experimental Results

Figure 4.5 and Figure 4.8 were previously published in [85]. The current strategy exploitability in Figure 4.1 was previously published in [9].

	CFR avg	CFR cur	CFR$^+$ avg	CFR$^+$ cur
HULHE			-1.66	-1.49
Rhode Island	-0.89	-0.22	-1.43	-0.91
Leduc	-0.72	-0.18	-1.33	-0.48
Kuhn	-0.80	-0.25	-0.84	-0.25
random	-0.75	-0.22	-1.64	-0.78
Blotto	-0.85	-0.22	-0.99	-0.26
RPS	-0.76	-0.24	-0.94	-0.25
pennies	-0.79	-0.27	-0.87	-0.25

Table 4.1: Convergence rates from log-log regression

	CFR avg	CFR cur	CFR$^+$ avg	CFR$^+$ cur
HULHE			-1.41	-1.48
Rhode Island	-0.85	-0.15	-1.50	-0.77
Leduc	-0.66	-0.33	-1.52	-0.31
Kuhn	-0.80	-0.26	-0.81	-0.27
random	-0.74	-0.23	-1.69	-0.72
Blotto	-0.83	-0.22	-0.95	-0.26
RPS	-0.77	-0.20	-0.93	-0.29
pennies	-0.81	-0.31	-0.83	-0.25

Table 4.2: Convergence rates from log-log regression in last 90%

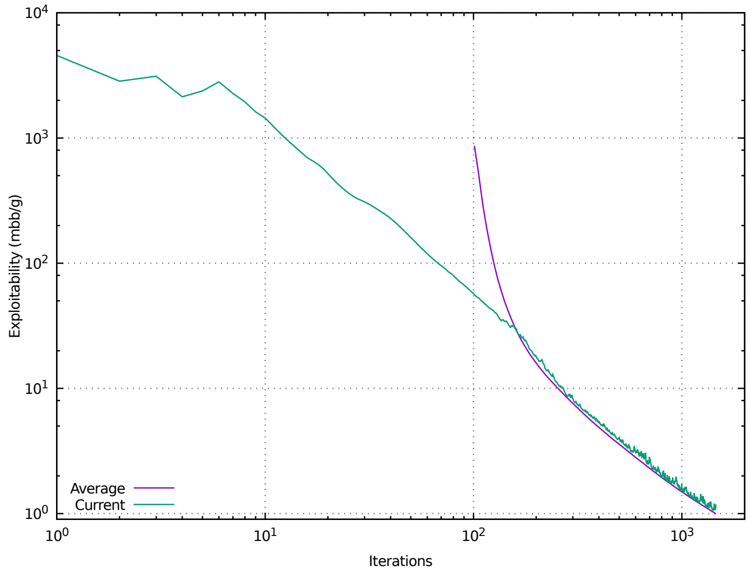


Figure 4.1: CFR$^+$ convergence rates in HULHE

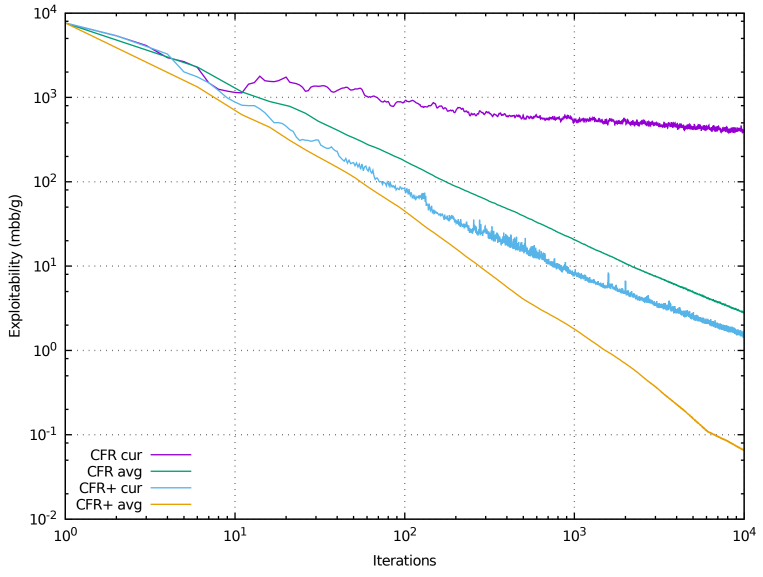


Figure 4.2: Convergence rates in Rhode Island hold'em

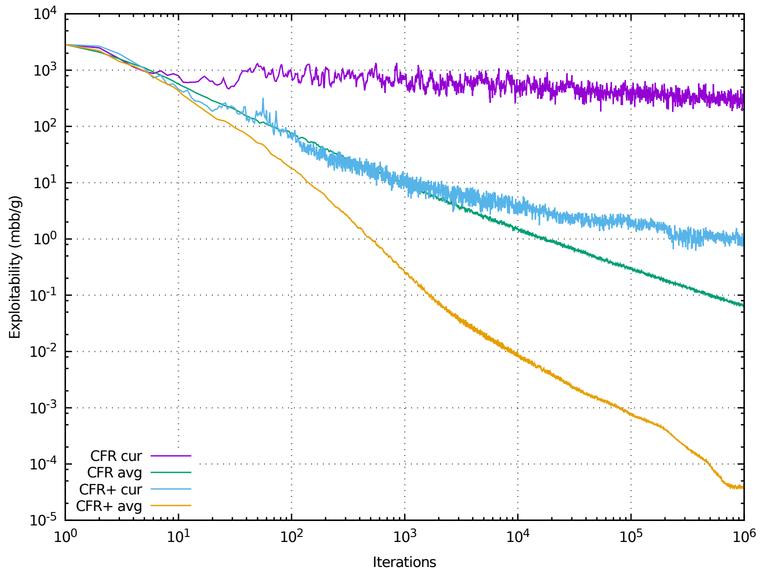


Figure 4.3: Convergence rates in Leduc hold'em

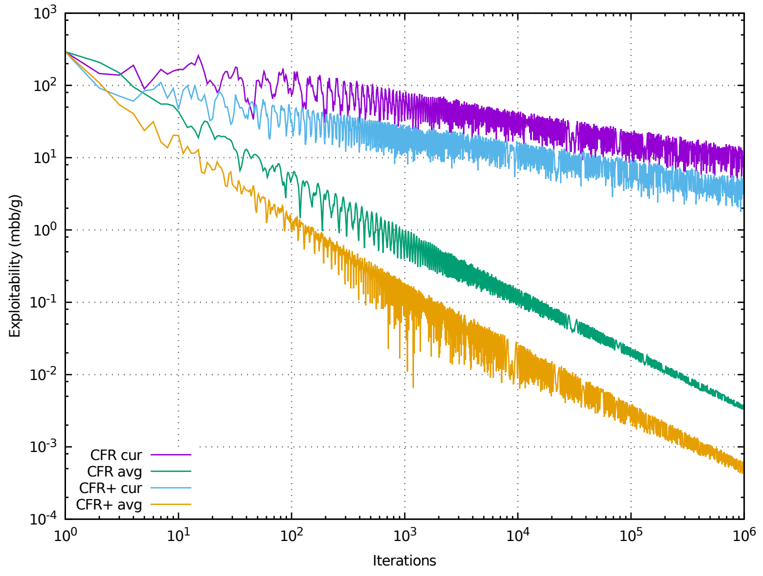


Figure 4.4: Convergence rates in 3 card Kuhn poker

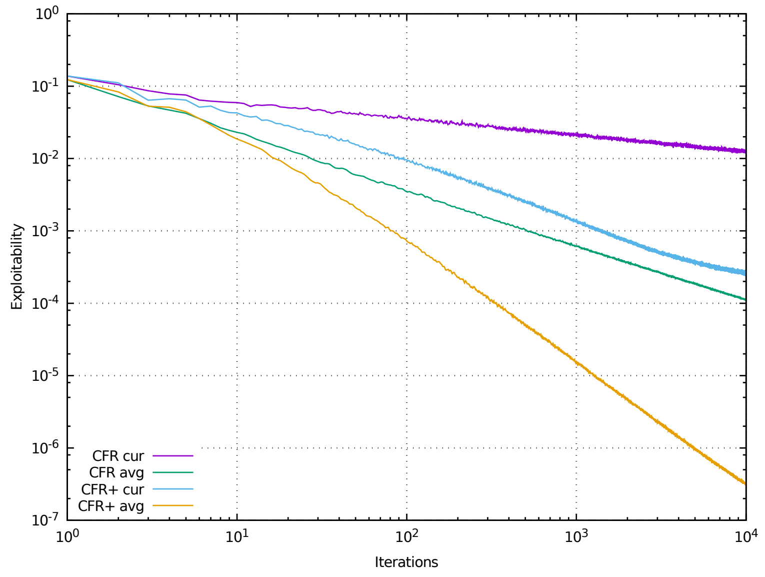


Figure 4.5: Convergence rates in 1000 $\times$ 1000 random matrix games

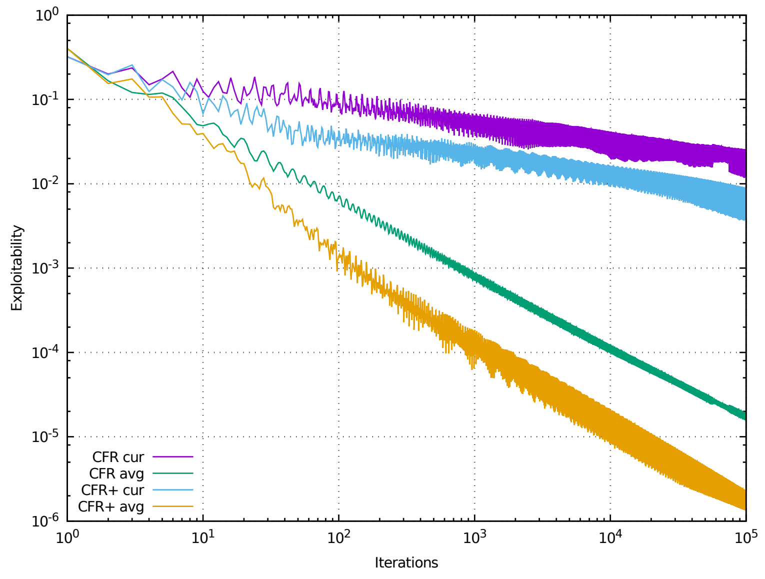


Figure 4.6: Convergence rates in 3,16 Blotto game

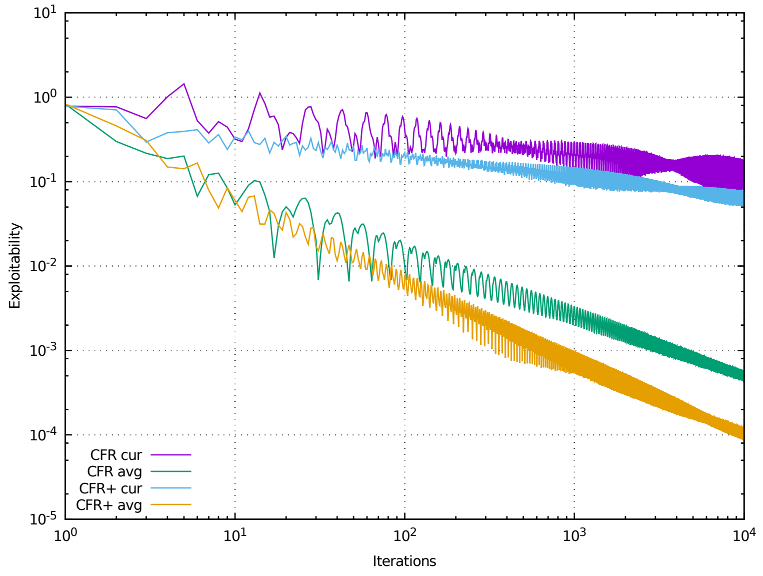


Figure 4.7: Convergence rates in rock-paper-scissors

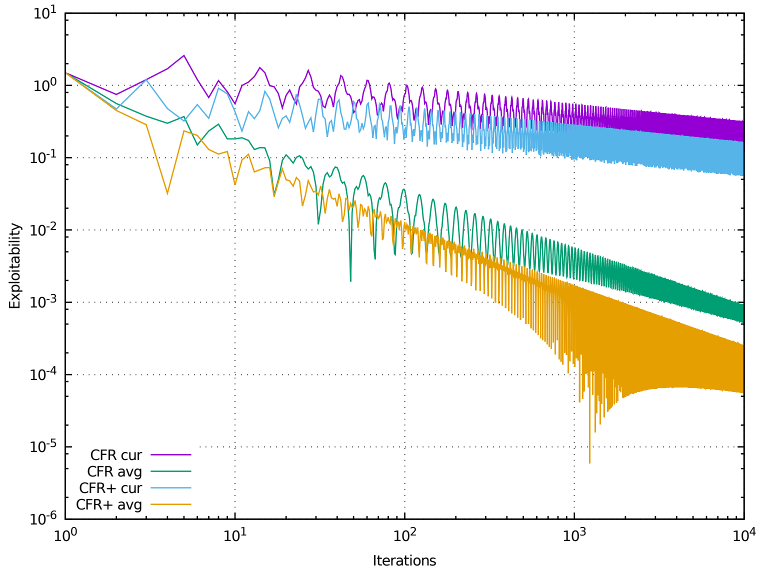


Figure 4.8: Convergence rates in the matching pennies game

4.3.3 CFR and CFR$^+$ Experimental Conclusions

The results in Section 4.3.2 strongly support the conclusion that in practice, CFR$^+$ is faster than CFR. In all games except rock-paper-scissors and the matching pennies game, the CFR$^+$ average strategy is always less exploitable than the CFR average after no more than 3 iterations. In rock-paper-scissors and matching pennies, the CFR$^+$ average strategy is usually less exploitable than the CFR average strategy, and is always less exploitable after somewhere between 100 and 1000 iterations.

Empirical rates of convergence for the average strategy are uniformly better (larger negative number) for CFR$^+$ than for CFR. For larger games like Rhode Island hold'em, Leduc hold'em, and 1000 by 1000 matrix games, CFR$^+$ proceeds almost quadratically faster. Given the rates for Rhode Island hold'em in Table 4.2 we might expect to need around 200 000 CFR iterations to do the work of 1000 CFR$^+$ iterations, while both Leduc hold'em and 1000×1000 matrix games would need more than 7 000 000 iterations.

We can say two things about the current strategy. First, the current CFR strategy improves very slowly, although it may be a surprise to see that it continues to improve at all. Second, there appears to be a correlation between the size of the game and the performance of the CFR$^+$ current strategy. It is only in the larger games that the CFR$^+$ strategy improved relatively quickly, with the best performance observed in HULHE, the largest game. While the choice to use the current strategy in Cepheus looked like a promising avenue to save time and space by avoiding the average strategy computation, this choice appears likely to be disastrously slower than using the average strategy.

4.3.4 Conjecture on CFR and CFR$^+$ performance

Despite the empirically observed difference in convergence rates for CFR$^+$ and CFR, I conjecture that the difference is transient, both algorithms have asymptotically equivalent behaviour, and the length of the transient period is determined by how long it takes to assign and retain a positive probability to the iteratively non-dominated support of the equilibrium. Note that this is not a claim with much relevance to the practical choice of algorithm: practically speaking, only the early performance is relevant, and CFR$^+$ outperforms CFR in that time.

The matching pennies game is so small that we can use it to make a few observations supporting the conjecture. The two differences between CFR$^+$ and CFR are the use of regret-matching$^+$ versus regret-matching, which affects the regrets and current strategy, and the use of linear versus constant weighting for the average strategy, which affects only the average strategy. Because of the asymptotically similar behaviour of the current strategy, we might expect the difference in the variability of the average strategy exploitability to be due to the averaging. We can verify this by running CFR with regret-matching$^+$, shown in Figure 4.9.

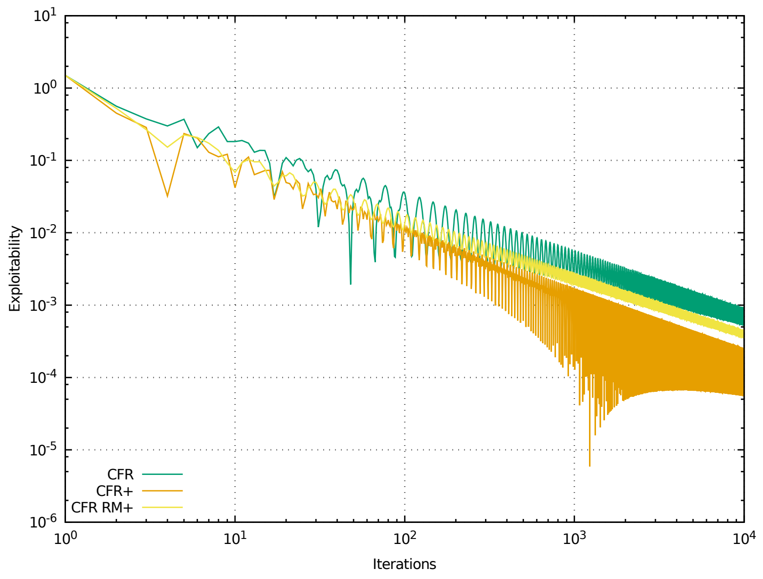


Figure 4.9: Exploitability over time for CFR, CFR$^+$, and CFR with regret-matching$^+$ in matching pennies game

If the difference in averaging is put aside as irrelevant, we are left with the difference between regret-matching$^+$ and regret-matching. Because there are only two actions for a player in the matching pennies game, we can plot the regrets for each of the row player's action, as shown in Figure 4.10.

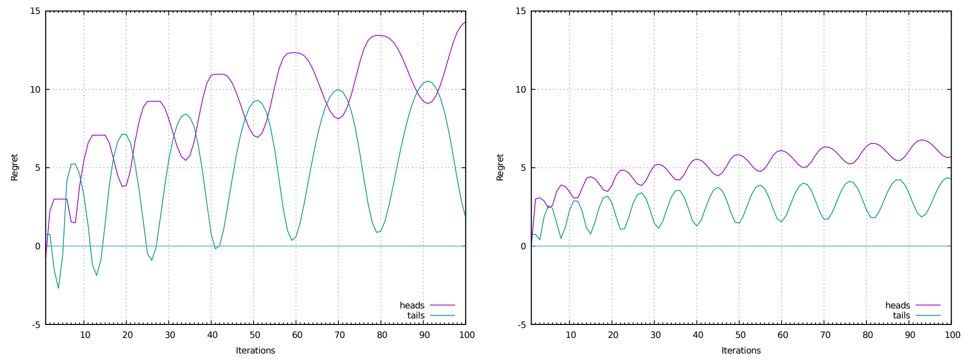


Figure 4.10: Regret over time for both row-player actions in matching pennies game, using CFR (left) and CFR$^+$ (right)

While the frequency and amplitude of the oscillations are different, the general behaviour is the same: oscillating regret values with a negative correlation between heads and tails, and an overall trend of increasing as the square root of the number of iterations. More importantly, after iteration 41, neither algorithm has negative regret for either action (for either player, not shown in Figure 4.10.) With only positive regrets, regret-matching and regret-matching$^+$ are identical. Starting at iteration 42, CFR and CFR$^+$ are continuing from different points, but generating the new current strategy with mathematically identical algorithms. If the difference in averaging is irrelevant and the algorithm for the current strategy is identical, the initial 41 iterations become irrelevant over time and both CFR and CFR$^+$ will have asymptotically equivalent behaviour.

If the maximum positive regret grows roughly as the square root of the number of iterations, we might expect that for all games, eventually the magnitude of any cycles in the regrets and strategies will be smaller than the minimum positive regret. The actions with $\mathcal{O}(\sqrt{T})$ positive regret must correspond to actions taken by strategies in the support of all Nash equilibria.

4.3.5 Weighted Average

Section 4.1.2 describes how the CFR$^+$ average strategy uses linearly increasing weights, rather the uniform weights used in CFR. While there is no strong theoretical justification for the linear average used by CFR$^+$, Figure 4.11 and

Figure 4.12 shows that CFR$^+$ does benefit from that choice in practice. By contrast, CFR does worse with the linear average than the uniform average.

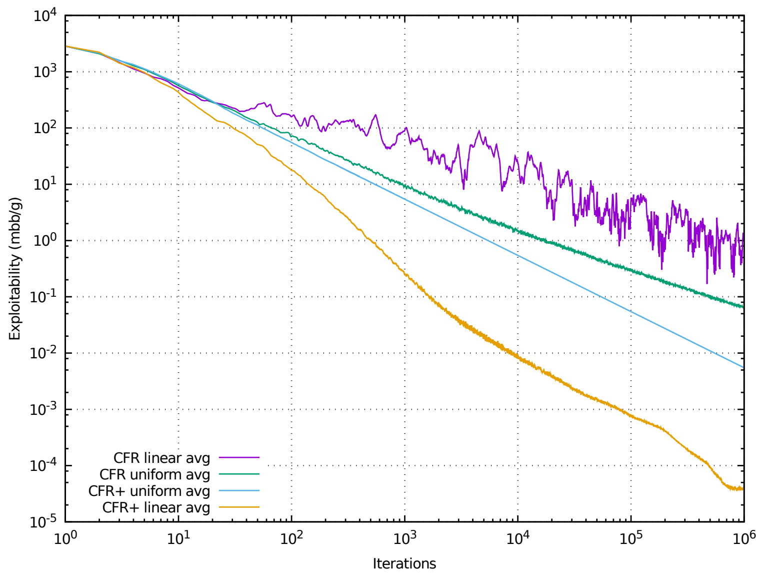


Figure 4.11: Convergence rates in Leduc hold'em with different methods for averaging strategies

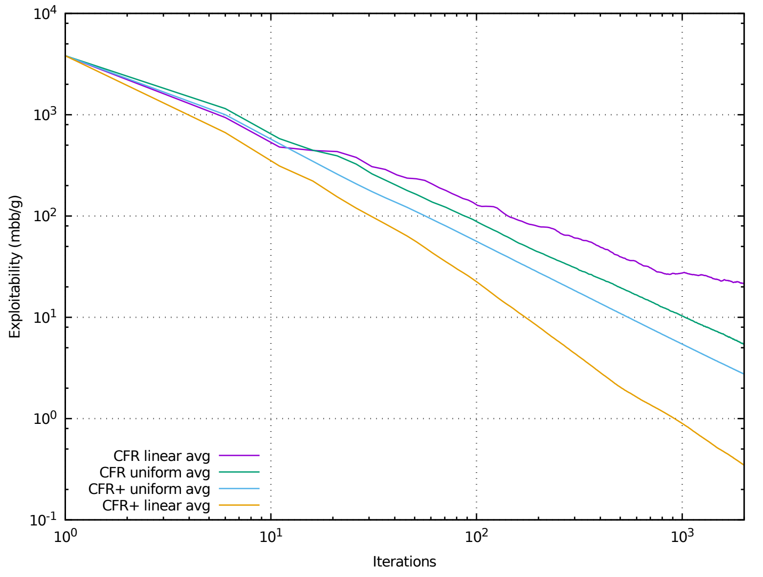


Figure 4.12: Convergence rates in Rhode Island hold'em with different methods for averaging strategies

4.3.6 Alternating Updates

As discussed in Section 4.1.3, CFR$^+$ explicitly states that the regrets should be updated in alternating fashion, whereas CFR is implicitly described as simultaneously updating both players but might be implemented using alternating updates. The use of alternating updates makes a very large difference for CFR$^+$, but still offers a substantial improvement in the convergence rate of CFR. Figure 4.13 shows the performance of CFR$^+$ and CFR with both alternating update and simultaneous update versions.

updates	CFR avg	CFR cur	CFR$^+$ avg	CFR$^+$ cur
alternating	-0.66	-0.33	-1.52	-0.31
simultaneous	-0.52	-0.02	-0.60	-0.24

Table 4.3: Convergence in Leduc hold'em with simultaneous updates, last 90%

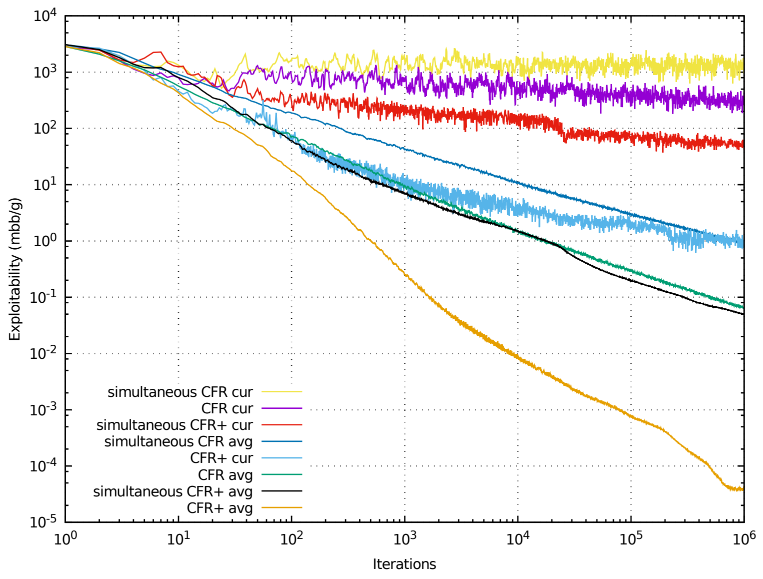


Figure 4.13: Convergence rates in Leduc hold'em with simultaneous updates

With simultaneous updates, CFR does no better than the $\mathcal{O}(1/\sqrt{T})$ upper bound described in Theorem 1, and CFR$^+$ only does as well as CFR with alternating updates. This performance difference between simultaneous and alternating updates means there is an additional hidden cost to using Monte Carlo variants of CFR or CFR$^+$. If each iteration is only updating a small portion of the tree, we are effectively doing simultaneous updates, even if we alternate between the players. For CFR, the lack of alternating updates is a small penalty that might be outweighed by the advantages of sampling. For CFR$^+$, the lack of alternating updates removes almost all of the empirically observed improvement in performance over CFR. A Monte Carlo variant of CFR$^+$ is unlikely to be effective in practice. We have tried and failed to produce such an algorithm that outperforms MCCFR.

4.4 Comparison to Accelerated Techniques

CFR$^+$ has good empirical performance, but its $\mathcal{O}(1/\epsilon^2)$ asymptotic guarantee is worse than the $\mathcal{O}(1/\epsilon)$ guarantee of an earlier method proposed by Gilpin *et al.* [31]. CFR$^+$ has the advantage of simplicity and no parameters that need to be tuned, and in this section I demonstrate that the performance is empirically competitive with, or better than, faster optimisation-based methods.

Trying to find a Nash equilibrium can also be looked at as a mathematical optimisation problem, letting us apply the large body of knowledge from that field. Of particular interest are the methods which are guaranteed to converge at a rate of $\mathcal{O}(1/\epsilon)$ when applied to non-smooth saddlepoint problems, as it improves on the $\mathcal{O}(1/\epsilon^2)$ rate of CFR and CFR$^+$. I will start with some background, applied to the simple case of mixed strategies (i.e. matrix games) and then consider sequence-form strategies.

4.4.1 Basic Optimisation Applied to Normal Form

Trying to find a player one strategy that is part of a Nash equilibrium in the space of mixed strategies is finding a solution to the mathematical optimisation problem $\min_{\mathbf{x}\in\Delta_m} \max_{\mathbf{y}\in Q} \mathbf{x}^\intercal A\mathbf{y}$, where $Q = \Delta_j = \{\mathbf{v}|\mathbf{v}\cdot\mathbf{1}^j = 1, \mathrm{v}_i \geq 0 \forall i\}$ is the j-simplex, m and n are the number of player 1 and player 2 pure strategies respectively, and A is an m by n matrix describing the player 2 utility. Because $\max_{\mathbf{y}\in\Sigma_2} \mathbf{x}^\intercal A\mathbf{y}$ is a convex function of $\mathbf{x}$ and $\mathbf{v} \in \Delta_j$ is a convex constraint, finding an optimal $\mathbf{x}$ is a convex optimisation problem.

To get an optimal player 2 strategy $\mathbf{y}$, we can swap $\mathbf{x}$ and $\mathbf{y}$, negate A, and solve the new optimisation problem. There are also methods, including the standard LP formulations, which simultaneously find both $\mathbf{x}$ and $\mathbf{y}$ by taking advantage of the fact that the objective is not just convex in $\mathbf{x}$, but is also concave in $\mathbf{y}$.

For very large problems, it becomes important to reduce space requirements. For this reason, first-order techniques for solving general convex optimisation problems are often used for solving large convex optimisation problems. These methods use only a few points, and a local computation of the gradient of the objective function to produce and use a linear lower bound model around the point.

One of the simplest first-order methods, the classical gradient descent algorithm, involves moving some distance in the direction of steepest descent as measured from the current point, and finding the nearest strategy. That is, with an objective function f, if we are at point $\mathbf{x}^t$ we generate a new point $\mathbf{x}^t - \eta^t \nabla f(\mathbf{x}^t)$, where η^t is a parameter controlling the distance. Strategy constraints on $\mathbf{x}$ are handled by doing a projection step $\operatorname{argmin}_{\mathbf{x}' \in Q} \|\mathbf{x}' - \mathbf{x}\|$ after the gradient step, where a commonly used choice for $\|\cdot\|$ is the Euclidean distance l_2 norm $\|\cdot\|_2$. Putting the two steps together, we get $\mathbf{x}^{t+1} = \operatorname{argmin}_{\mathbf{x}' \in Q} \|\mathbf{x}' - (\mathbf{x}^t - \eta_t \nabla f(\mathbf{x}^t))\|$.

With fixed η the sequence of points generated by gradient descent is guaranteed to converge towards an optimum if f is not just convex, but $L-$smooth: $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|^* \leq L\|\mathbf{x} - \mathbf{y}\|$ for some $L > 0$, where $\|\mathbf{z}\|^* = \max_{\mathbf{x}, \|\mathbf{x}\| \leq 1} \langle \mathbf{z}, \mathbf{x} \rangle$ is the dual norm of $\|\cdot\|$. In this case, we get a quadratic upper bound $f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + L/2\|\mathbf{y} - \mathbf{x}\|^2$ which we can minimise to maximise the guaranteed progress $\mathbf{x}^{t+1} = \operatorname{argmin}_{\mathbf{x}' \in Q}(\langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + L/2\|\mathbf{x}' - \mathbf{x}^t\|^2)$. For the L^2 norm, this corresponds to a step size of $1/L$. Gradient descent on smooth functions has a convergence rate of $\mathcal{O}(1/\epsilon)$.

For non-smooth functions where the derivative may not be defined at $\mathbf{x}$ we must use subgradients, a linear lower bound ∂f on the function such that $\forall \mathbf{x}'$, $f(\mathbf{x}') \geq f(\mathbf{x}) + \partial f \cdot (\mathbf{x}' - \mathbf{x})$. We must also use a different scheme for selecting the step size, either by selecting η as function of the desired error, or by using a decreasing sequence η_t resulting in a convergence rate of $\mathcal{O}(1/\epsilon^2)$ [80, 35].

There is another class of $\mathcal{O}(1/\epsilon^2)$ methods for directly solving non-smooth functions, working within the dual space of the problem. Dual-descent algorithms use a prox function $d(\mathbf{x})$ [74] (also known as a distance generating function) to map back and forth between the primal and dual spaces, and handle projection in the primal space by using the associated distance-like Bregman divergence $D(\mathbf{x}, \mathbf{y}) = d(\mathbf{x}) - d(\mathbf{y}) - \langle \nabla d(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle$ [11]. The prox function (and therefore the associated Bregman divergence) must be continuous, strongly convex functions (relative to the chosen norm $\|\cdot\|$.)

Two simple first-order dual-descent methods are Mirror Descent [69] and Dual Averaging [72]. Both methods produce successively better linear lower bound estimates by averaging the gradient at a sequence of points, so that $\bar{\mathbf{x}}^t$ converges to the minimum, in contrast to $\mathbf{x}^t$ converging in primal methods. Mirror Descent uses a step size η and picks

$$\mathbf{x}^{t+1} = \operatorname*{argmin}_{\mathbf{x}' \in Q}(D(\mathbf{x}', \mathbf{x}^t) + \langle \eta \nabla f(\mathbf{x}^t), \mathbf{x}' - \mathbf{x}^t \rangle)$$

while the Dual Averaging scheme uses a sequence of weights η_t and picks

$$\mathbf{x}^{t+1} = \operatorname*{argmin}_{\mathbf{x}' \in Q}(d(\mathbf{x}') + \sum_{t'=1}^{t} \langle \eta_i \nabla f(\mathbf{x}^{t'}), \mathbf{x}' - \mathbf{x}^{t'} \rangle)$$

In order to be tractable, the prox function must be picked so that the minimisation problem at each step can be efficiently computed. For example, using the l_1 norm over $\mathbf{R}^N$ with a constraint set Δ_n, the negative entropy function $d(\mathbf{x}) = \log(n) + \sum_{i=1}^{n} \mathrm{x}_i \log(\mathrm{x}_i)$ and its associated KL divergence function $D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mathrm{x}_i \log \frac{\mathrm{x}_i}{\mathrm{y}_i}$ are strongly convex. The corresponding Mirror Descent step $\mathbf{x}^{t+1} = \operatorname{argmin}_{\mathbf{x}' \in Q}(D(\mathbf{x}', \mathbf{x}^t) + \langle \eta \mathbf{v}, \mathbf{x}' - \mathbf{x}^t \rangle)$ has a closed form solution $\mathrm{x}_i^{t+1} = \mathrm{x}_i^t e^{\eta \mathrm{v}_i} / \sum_{j=1}^{n} \mathrm{x}_j^t e^{\eta \mathrm{v}_j}$, giving the Hedge update rule.

4.4.2 Fast Optimisation

Extending on work showing a fast $\mathcal{O}(\sqrt{1/\epsilon})$ first-order method for smooth objective functions, Nesterov introduced a fast $\mathcal{O}(1/\epsilon)$ method for optimising a class of non-smooth functions [71]. Given an objective function that is a maximum over some space, this method uses a smooth approximation of the objective function, generated by adding a continuous strongly convex function weighted by a smoothness parameter (the same conditions required for a prox function.) As with the per-iteration minimisation problem involving the Bregman divergence in dual-descent methods, Nesterov's method and later variants [87] require an almost-identical optimisation problem at each step, so the norm and smoothing function must be chosen carefully to get a tractable problem.

While these methods are sufficient to solve saddlepoint problems like finding a game equilibrium, following Nesterov's initial results there were methods developed which specifically target saddlepoint problems. For example, Nesterov introduced the excessive gap technique [70] and Nemirovski introduced a prox-method for saddlepoint problems [68].

4.4.3 Optimisation Applied to Sequence Form

There are two issues that arise when moving from normal form games to extensive form games. First, we note that we cannot directly optimise for behaviour strategies in an efficient way. The space of behaviour strategies is convex (a set of simplex constraints at each information set), but the objective function is not (involving products of variables.) We can, however, consider optimisation using the sequence form strategy representation introduced by Koller *et al.* [51].

Using sequence form, we get a problem similar to the normal form optimisation, $\min_{E\mathbf{x}=\mathbf{e}} \max_{F\mathbf{y}=\mathbf{f}} \mathbf{x}^{\intercal} A\mathbf{y}$, where the constraints on $\mathbf{x}$ and $\mathbf{y}$ enforce the tree structure of the sequences, m and n are the number of player 1 and player 2 terminal sequences respectively, and A is an $m \times n$ matrix describing the player 2 utility. Because the objective and constraints are still convex, this sequence form version is another convex optimisation problem, much smaller than the mixed strategy problem. This optimisation problem, in a modified form more suitable for LPs, is the basis of the algorithm of Koller *et al.* [51].

The second issue is that all of the optimisation methods for non-smooth problems discussed require a prox function, and the ability to efficiently solve projection problems of the form $\mathrm{argmin}_{\mathbf{x}' \in Q}(D(\mathbf{x}', \mathbf{x}^t) + \langle \eta \nabla f(\mathbf{x}^t), \mathbf{x}' - \mathbf{x}^t \rangle)$ at each iteration. Standard choices for d/D like negative entropy/KL divergence are still strongly convex in Q, but the minimisation can no longer be computed efficiently, let alone have a closed form solution.

Hoda *et al.* solved this problem by introducing a recursively constructed prox function for sequence form problems, based on a choice of a base prox function at each information set [38]. The recursion builds from the root of the game to the leaves, with $\psi_i(\mathbf{x}) = d(\mathbf{x}_i) + \sum_{\mathbf{x}_{i+1}} p(\mathbf{x}_{i+1})\psi_{i+1}(\mathbf{x}_{i+1}/p(\mathbf{x}_{i+1}))$ being the base prox function at root information set at level i plus the recursive construction at each level $i+1$ subtree $\mathbf{x}_{i+1}$, scaled by the value of the parent sequence $p(\mathbf{x}_{i+1}) \in \mathbf{x}_i$. Hoda *et al.* also give an efficient recursively defined solution for the projection problem which frequently occurs in optimisation.

Gilpin *et al.* used the recursively constructed prox function of Hoda *et al.* to apply Nesterov's excessive gap technique to poker-like games to find a Nash equilibrium profile in $\mathcal{O}(1/\epsilon)$ time [31]. They considered prox functions based on both negative entropy and Euclidean distance, and found the entropy based prox function to be the better choice.

4.4.4 Connection to and Comparison with CFR$^+$

If we consider the Bregman divergence $\Phi(\mathbf{x}', \mathbf{x})$ of the recursive prox function $\psi(\mathbf{x})$ generated from a prox function $d(\mathbf{x})$ with Bregman divergence $D(\mathbf{x}', \mathbf{x})$, we get $\Phi_i(\mathbf{x}', \mathbf{x}) = D(\mathbf{x}_i) + \sum_{\mathbf{x}_{i+1}} p(\mathbf{x}'_{i+1})\Phi(\mathbf{x}'_{i+1}/p(\mathbf{x}'_{i+1}), \mathbf{x}_{i+1}/p(\mathbf{x}_{i+1}))$. That is, the divergence has the same recursive structure as the prox function. If we consider using mirror descent with $D(\mathbf{x}', \mathbf{x})$ being KL divergence, we get an update which resembles CFR using Hedge. There are two differences. First, instead of using the value from our children from the current strategy, we use the new strategy. Second, we add the subtree's distance function to the returned value, giving the algorithm a preference for actions where the subtree strategy has not changed.

While regret-matching does not correspond to a prox or divergence function, we can construct new regret-matching algorithms through analogy from theoretically sound optimisation methods, by replacing Hedge-like operations with regret-matching or regret-matching$^+$ operations. We cannot easily copy the preference of actions leading to subtrees with less change, but we can modify CFR and CFR$^+$ to use the new strategy rather than the current strategy, as mentioned in Section 4.1.5. As shown in Figure 4.14 this change has a small positive effect, and is used in the experimental results in this chapter.

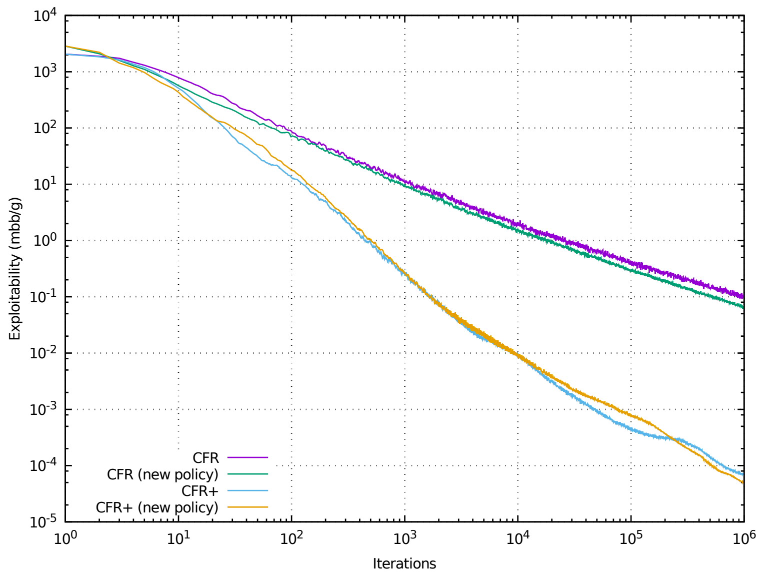


Figure 4.14: Convergence rates in Leduc hold'em when updating parent regrets using the current strategy and the new strategy

There have been many different $O(1/\epsilon)$ methods developed [87] since the introduction of Nesterov's initial accelerated method for non-smooth convex optimisation. I chose three simple accelerated algorithms for comparison to CFR$^+$: a version of Nesterov's algorithm labeled as Algorithm 1 in the survey by Tseng [87] using Nesterov's suggestion for a smoothed approximation of $\max_{\mathbf{y}} \mathbf{x}A\mathbf{y}^\intercal$ [71], Nemirovski's mirror prox saddlepoint method [68], and a more recent technique that uses "optimistic" regrets to directly modify a self-play regret minimisation algorithm [73, 83].

Along with CFR, CFR$^+$, and the three optimisation-based algorithms, I also use two theoretically unjustified algorithms constructed from the optimisation based algorithms: Nemirovski's saddlepoint method with regret-matching$^+$, and optimistic regret updates with regret-matching$^+$. I do not include the two variants using regret-matching, as the results are virtually identical to the regret-matching$^+$ variants. I do not consider regret-matching$^+$ or regret-matching versions of Algorithm 1 from Tseng's survey, as both variants fail to converge.

All of the experiments in this section will look at the change in exploitability over increasing iterations. This is a slightly misleading measurement, as the time per iteration varies over the different algorithms by some constant, but it was not reasonable to generate separate high performance implementations for all algorithms. Roughly speaking, the optimistic regret method has the same cost per iteration as CFR or CFR$^+$, and both Nemirovski's saddlepoint method and Algorithm 1 in Tseng's survey do roughly twice the work per iteration. As implemented, the Hedge variants are roughly ten times slower than regret-matching or regret-matching$^+$. While CFR and CFR$^+$ store two values per sequence, all the other algorithms store three values per sequence.

Figure 4.15 and Table 4.4 shows the behaviour of the seven algorithms in Leduc hold'em, and Figure 4.16 looks at the behaviour of the current strategy. Figures 4.17 and 4.18 and Table 4.5 shows the behaviour in Kuhn hold'em.

I draw two conclusions from these results. First, CFR$^+$ is competitive with accelerated algorithms. Looking at the average strategy, which all algorithms use as their prescribed output, CFR$^+$ significantly outperforms all other algorithms in Leduc hold'em, and is faster at reaching any target exploitability up to 10^{-2} mbb/g in Kuhn poker (a hundred-thousandth of a chip.) Combined with the results in Table 4.1, which show a rough trend of better CFR$^+$ performance in larger games, it is reasonable to say that CFR$^+$ is a competitive choice for finding ϵ-Nash equilibria in large games.

Second, the current strategy exhibits interesting behaviour when using Nemirovski's saddlepoint method, the optimistic regret method, and the regret-matching$^+$ variants of both algorithms. After some number of iterations, the exploitability of the current strategy appears to drop geometrically, rather than as T^{-k} for some fixed k. For example, in Kuhn poker, after 125 iterations the current strategy for optimistic regret-matching$^+$ is a 10^{-16}-Nash equilibrium. If a very low exploitability is required, it may be worth some hybrid method which combines the fast initial progress of CFR$^+$ with a fast optimisation method, watching for a phase change in the behaviour of the current strategy.

Algorithm	Convergence Rate	Conv. Rate Last 90%
CFR	-0.72	-0.66
CFR$^+$	-1.33	-1.52
Tseng Alg. 1	-0.84	-0.87
Nemirovski (both variants)	-1.0	-1.0
Optimistic (both variants)	-1.0	-1.0

Table 4.4: Convergence rates in Leduc hold'em for multiple algorithms

Algorithm	Convergence Rate	Conv. Rate Last 90%
CFR	-0.80	-0.80
CFR$^+$	-0.84	-0.81
Tseng Alg. 1	-0.94	-1.04
Nemirovski (both variants)	-1.0	-1.0
Optimistic (both variants)	-1.0	-1.0

Table 4.5: Convergence rates in Kuhn poker for multiple algorithms

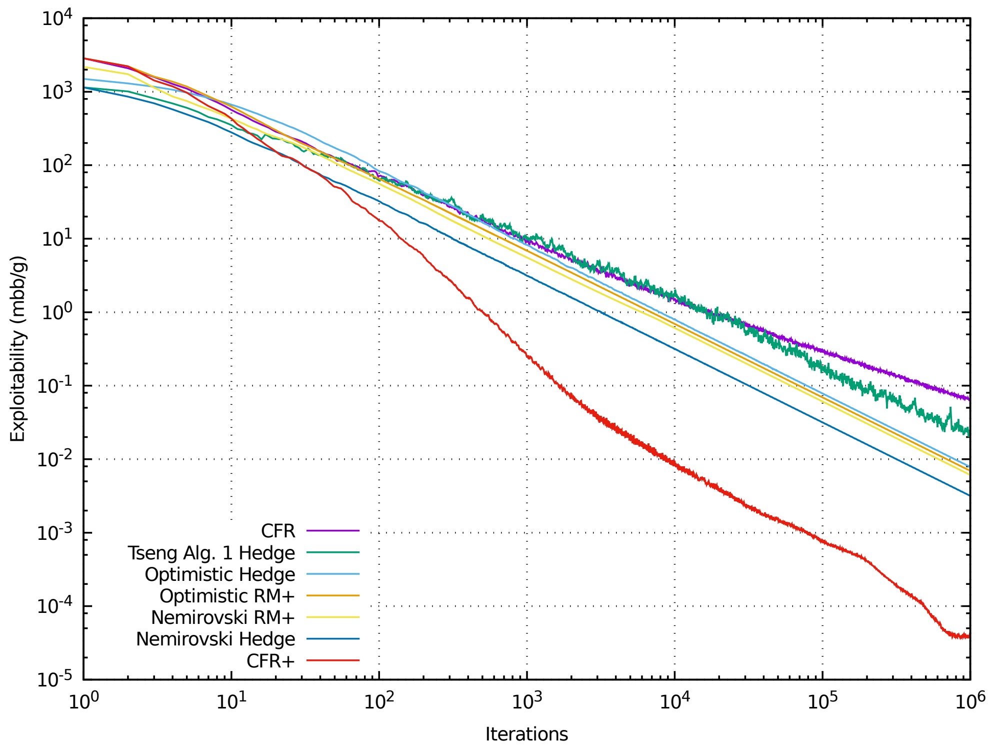


Figure 4.15: Convergence rates in Leduc hold'em for multiple algorithms

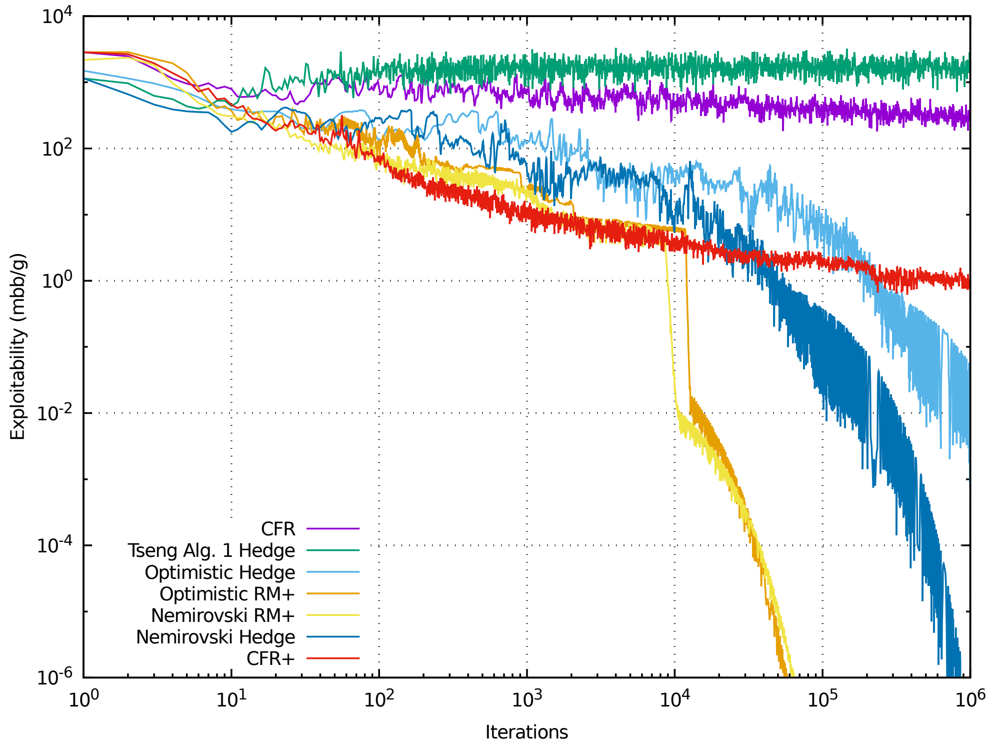


Figure 4.16: Current strategy behaviour in Leduc hold'em for multiple algorithms

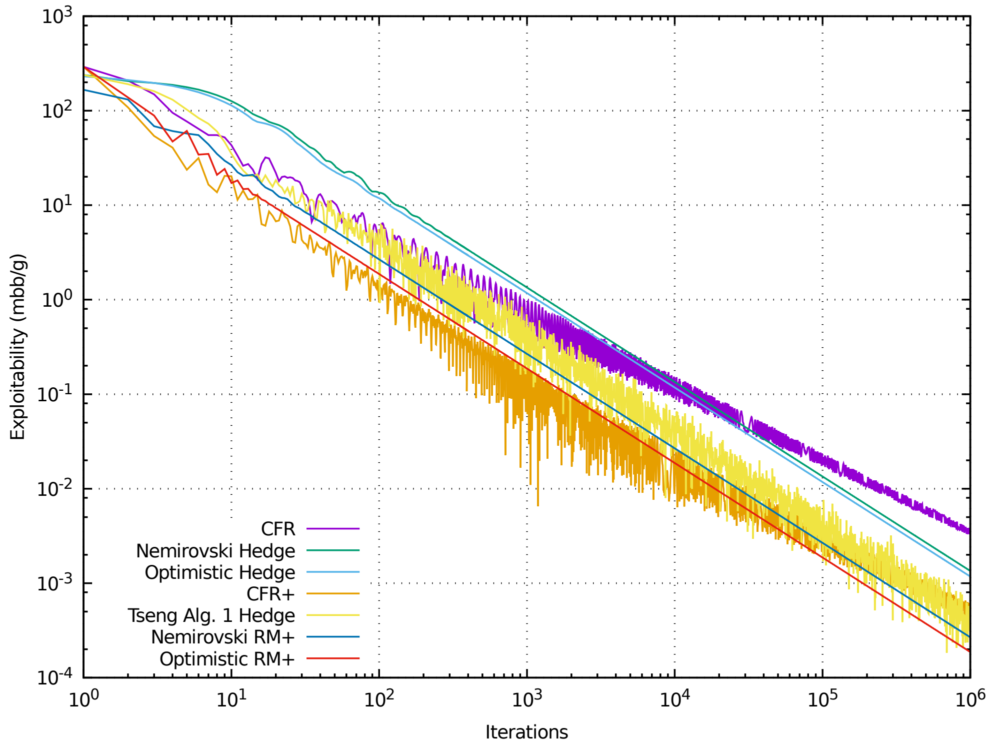


Figure 4.17: Convergence rates in Kuhn poker for multiple algorithms

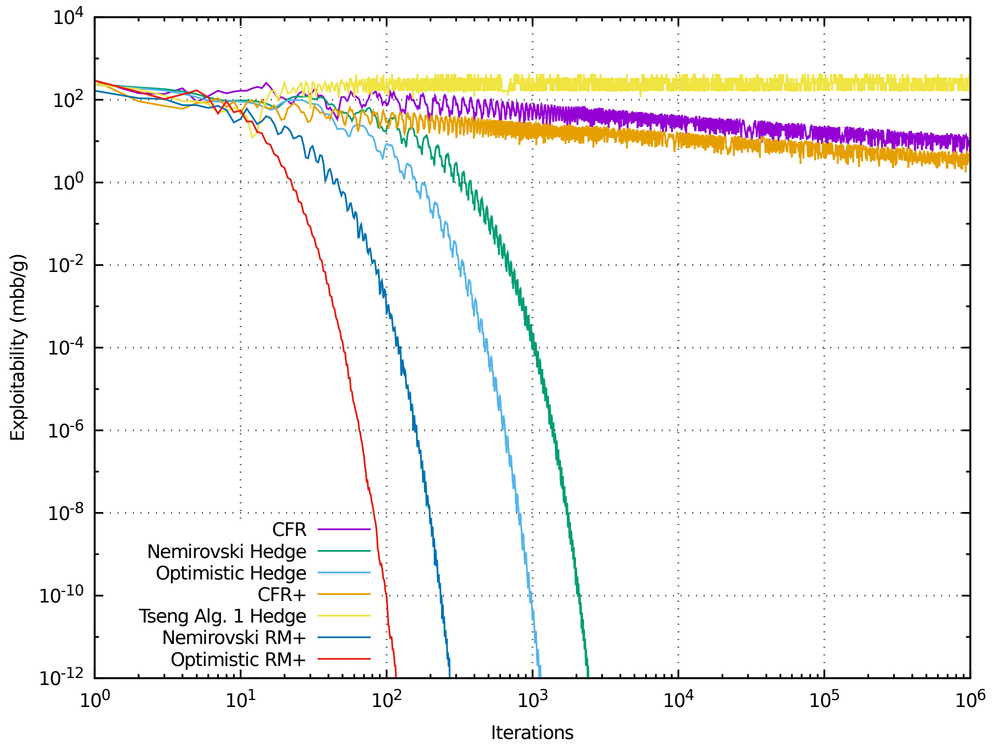


Figure 4.18: Current strategy behaviour in Kuhn poker for multiple algorithms

4.5 Monte Carlo CFR^{+}

The family of MCCFR algorithms introduced by Lanctot *et al.* [59] uses Monte Carlo sampling to update a small subset of information sets at each iteration. With a careful choice of sampling schemes, the update is both simple and efficient. For example, the external-sampling scheme described by Lanctot *et al.* samples opponent and chance events according to the current strategy, and has an update rule that is arguably less complex than CFR. At the same time, with high probability, external-sampling MCCFR has the same asymptotic behaviour over iterations, but has asymptotically faster iterations.

MCCFR algorithms also have the benefit that they work well with abstraction. If we use a function to map real game states to abstract game states, we can sample abstract states by sampling real states and then apply the mapping function. In contrast, for CFR we need to compute exact chance probabilities and leaf values for the abstract states. To avoid traversing the real game at every iteration – exactly what we tried to avoid by using abstraction – we must precompute and store those probabilities and values, which can be a non-trivial storage overhead.

We might naturally consider a new MCCFR$^+$ algorithm, using the same changes from CFR to CFR$^+$, to get the benefits of sampling and CFR$^+$. There are no technical difficulties switching MCCFR from regret-matching to regret-matching$^+$, and we could still use linear weighting when computing the average strategy. Unfortunately, MCCFR$^+$ is outperformed by MCCFR.

Looking at the behaviour of MCCFR$^+$, we can see a number of problems. First, regret-matching$^+$ sets $Q^t = \max(0, (Q^{t-1} + \Delta R^t))$, so variance in the value estimates may push the regrets above 0 for actions with a low average value. Second, the progress per iteration seems to be too slow to justify a linear weighting for the average strategy. MCCFR$^+$ is not rescued by small fixes like using a lower threshold than 0 for regret-matching$^+$, and using $\sqrt{t}$ weighting or uniform weighting for the average. Improving the performance of MCCFR$^+$ only seems to bring it up to par with MCCFR.

There may be a less obvious set of changes that make MCCFR$^+$ work, but the similarity in behaviour of CFR$^+$ and accelerated techniques suggests that a fix might not exist. Devolder *et al.* show that in a setting where there are inexact values, an accelerated method will accumulate errors over the iterations [22], and “slow” algorithms may have better performance. Sampling a subset of leaves does introduce error into the value computations, so there may be no sampling algorithm which is asymptotically faster than MCCFR.

Chapter 5

Using Less Space: Decomposition

There is a common factor in most algorithms for perfect information games that is missing in most algorithms for imperfect information games. Perfect information games are easy to decompose: we can split them into parts. Traditionally, imperfect information games have been considered a generally indivisible whole. The distinction is important, because decomposition can be used to reduce the space requirements of an algorithm, and can be combined with value estimates to trade reduced solution quality for reduced computation time.

Using algorithms which rely on decomposition[1] we have seen super-human play in games like chess (with 10^{40}-10^{50} reachable states [78]) and recently in the game of go [81] (with somewhat less than 10^{171} reachable states [86].)

For imperfect information games, while abstraction can let us solve a hopefully similar game, directly applying general purpose game solving algorithms limits us to games where the number of information sets is on the same order of size as available memory (*e.g.,* with roughly ten TB of available memory we can solve a game with around 10^{14} information sets, if we compress the stored regret values [9].) If space is tight, using decomposition to get around memory limitations is very appealing. Early game-theoretic approaches to HULHE

[1]Computing the value or the optimal action at a state without solving the entire game requires decomposition. Note that the ability to split the game into parts is not seen as novel or worthy of note, but rather is part of the accepted background of techniques for perfect information games.

used unsound, theoretically unjustified decomposition methods [4] to split the game into 9 pieces, because at the time the alternative would be to not even try solving the game due to lack of space.

My contributions are some fundamental tools for decomposition in imperfect information games, an algorithm CFR-D which uses decomposition to solve games, an algorithm for re-solving a portion of a game (all published in AAAI [15]), and some unpublished enhancements to the re-solving algorithm. CFR-D and re-solving trade decreased space requirements for increased computation time, letting us store fewer values than there are information sets. While solving an imperfect information game the same size as go would remain computationally intractable, it would now be a computation that we could run, even if it didn't finish in our lifetime: reducing the space requirement is a fundamental difference when the game size is larger than the number of particles in the observable universe.

5.1 Background: Decomposition and Perfect Information Games

I will start by explaining decomposition in perfect information games, both because perfect information games are a less complicated domain to introduce the concepts, and to contrast this ease with the difficulties encountered in imperfect information games.

We need a number of ideas and tools to use decomposition to solve games. We need to define the trunk and subgame, the parts we get when we split up a game. There must be some way to summarise a strategy in a subgame. We must have a way to find the correct trunk strategy, using summaries of subgame strategies. Finally, given a trunk strategy we need to be able to find the appropriate subgame strategy to get a complete strategy for the game.

5.1.1 Subgames and Trunk

Decomposition involves splitting a game into parts, so we must define those parts. A subgame is defined to be a state and all of its children. Put another way, the root of a subgame is a state, and the subgame is a set of states closed under the relationship "is a descendant of". That is, if a state s is in the subgame, all states s' which are descendants of s must be in the subgame. This definition is sometimes extended to consider a set of states at the root, and the descendants of all root states.

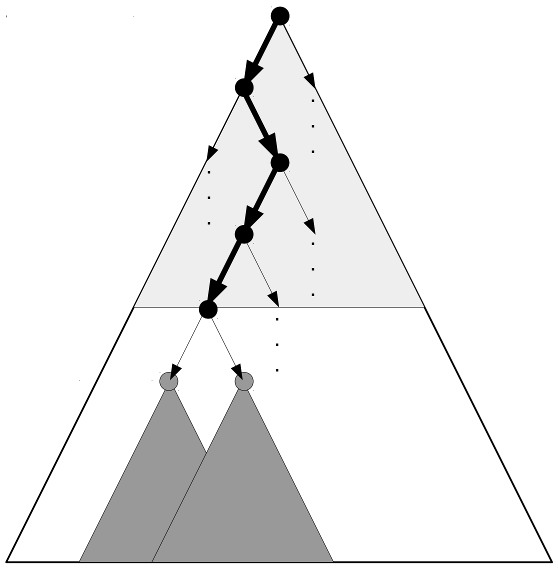


Figure 5.1: Perfect Information Game Tree with Subgames

The trunk of a game can be defined in two different, equivalent ways. First, given a game and a set of subgames, the trunk can be defined through omission as all of the states that are not in a subgame. Second, we could define the trunk as some set of states starting at the beginning of the game, such that for any state in the trunk other than the beginning of the game, its parent must also be in the trunk. The subgames are implicitly defined, with a separate subgame rooted at every state which is not in the trunk, but has its parent state in the trunk. In either case, the trunk and subgames partition the set of of states.

Consider the game tree depicted in Figure 5.1, where players have a sequence of binary decisions. By convention, the whole game tree is represented by a triangle, with the start of the game at the top. In this case, we also display a sequence of states represented by circles, and action choices represented by arrows, with the chosen action being highlighted with a thicker arrow. The dark shaded triangles are both subgames: some root state, and all states that can be reached from the root. There are also 30 other subgames which we do not show in the figure, which are reached by making different action decisions in the light shaded triangle which depicts the trunk. In summary, there are 31 states in the trunk, 32 subgames, and if there are 3 binary decisions in each subgame, then each subgame has 15 states.

If we are using behaviour strategies, which give a probability distribution over actions for each state (perfect information games have information sets with single states), a subgame or trunk strategy is a probability distribution over actions for all states in the subgame or trunk, respectively. Because the subgames and trunk partition the game, we can combine a strategy for the trunk and all subgames to get a strategy for the whole game. It can sometimes also be useful to consider modifying a full game strategy σ by using a new subgame strategy σ_S, in which case we replace the probability distributions of σ with σ_S for all states in the subgame.

Note that we can apply decomposition recursively. By the definition of a subgame, the whole game can be considered a subgame. Similarly, a subgame can be considered as defining a separate problem, which we could decompose into a trunk and subgames.

5.1.2 Trunk and Subgame Strategies

When we are trying to find an optimal strategy for a game, we can summarise our strategy within a subgame using a single value. Specifically, we need to know the value we would expect to receive upon reaching that subgame, given that we acted according to our strategy, and the opponent acted with a best response. Once we have computed that value, we can discard our subgame strategy, letting us save whatever space would have been needed to remember the strategy.

Summarising a subgame strategy with our value against a best response is sufficient to compute the strategy in the trunk. Returning to Figure 5.1, if we are trying to decide the best action to make at the final state (black circle) in the trunk (light shaded triangle), we only need to know whether the left subgame (dark shaded triangle) gives us a better value or the right subgame gives us a better value. The maximum of these two subgame values would, in turn, be the value of a subgame rooted at that final trunk state, which could be used in the computation of the optimal action earlier in the trunk.

5.1.3 Re-solving a Subgame

If we already have a trunk strategy, and we wish to know how to play within a subgame, we can use whatever algorithm we used to find the optimal subgame strategy when computing the trunk strategy. In perfect information games, it is irrelevant how likely we are to get to a subgame, or what either player's strategy is elsewhere in the game.

5.1.4 Putting the Tools Together

In practice, decomposition in perfect information games is often done recursively, in the maximal fashion: the game is split up so that the trunk is the single root state[2], with each of the subtrees being treated as games which are themselves split up with a single trunk state. For example, Alpha-Beta search [36] is recursive algorithm for finding the optimal move at the current state, with some pruning rules that allow to avoid considering entire subtrees.

If we ignore performance enhancements of various search algorithms, most algorithms for perfect information games start from some current state, whatever it is, compute (or approximate) an optimal strategy for the subgame, and only remember the next move. After making that move and any opponent moves in response, we start again solving from some new state. For these fully recursively decomposed algorithms, storage requirements can be drastically reduced to be linear in the remaining depth of the game. If we are okay with approximations of unknown quality, we can also drastically reduce com-

[2]In some cases we may have an opening book of moves for the beginning of the game, which could be thought of as the strategy in a larger trunk with multiple states.

putation time by limiting the search depth and stopping the recursion early, and replacing the computation after that depth with a heuristic evaluation function.

5.2 Decomposition in Imperfect Information Games

To make the conceptual tools of decomposition work within imperfect information games, we must re-work the way we define subgames, solve subgames, solve the trunk, and re-solve subgames to get a complete strategy. All of the work in this section was published in AAAI 2014 [15].

5.2.1 Trunk and Subgames

The traditional definition of a subgame deals with single states and their descendants, which can lead to impossible strategies in imperfect information games. The problem is that a strategy in an imperfect information game is defined on information sets, not states.

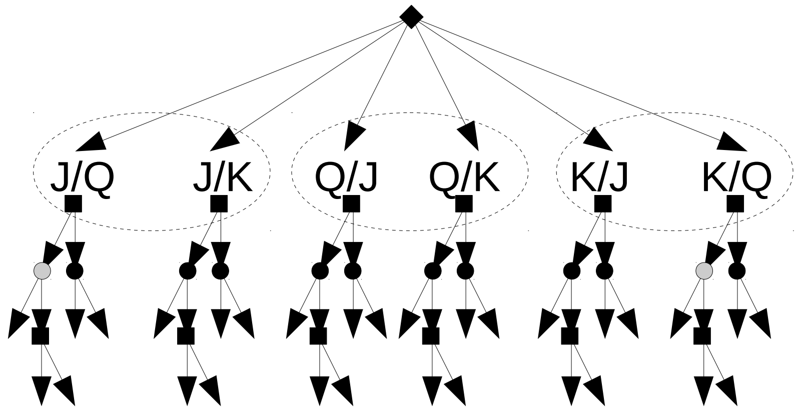


Figure 5.2: Partial Depiction of the Kuhn Poker Game Tree

For example, consider the 3 card game of Kuhn poker, where each player is dealt a single private card from a deck of J, Q, and K. There are six possible ways the cards can be dealt to the two players, followed by some player betting actions, as visualised in Figure 5.2. This means there are also six possible states where the first players makes their first action. There are, however, only three information sets, indicated by the dashed ovals (later information sets are not indicated.) The first player cannot distinguish between Q/J and Q/K (or J/Q and J/K, or K/J and K/Q), because they only differ in the second player's card, which the first player does not see. If we were to consider the strategy in the Q/J subgame and Q/K subgame independently, we might come to the conclusion that we should bet with Q/J and check with Q/K, but it is not possible for the first player to distinguish these situations and follow that strategy.

In order to guarantee that a subtree strategy is plausible in the full game, we need to guarantee that the subtree does not split up any information sets. We define an *imperfect information subgame* with this additional requirement. The root of an imperfect information subgame is now a set of states, and the subgame has a set of states which is closed under both the "is a descendant of" and the "is in the same information set" relationships. That is, if state s is in the subgame and I is the information set such that $s \in I$, then all descendants of s and all states in I are in the subgame.

In the game of Kuhn poker, shown in Figure 5.2, if the trunk is the single initial pre-deal state, all 6 states after dealing the cards must be in a single imperfect information subgame. The initial J/Q and J/K states must be in the same imperfect information subgame, because they are in the same information set. The initial J/Q and K/Q states must be in the same imperfect information subgame, because J/Q and its descendants must be in the same subgame, K/Q and its descendants must be in the same subgame, and for states where player two acts (un-filled arrows) J/Q and K/Q are indistinguishable, so the two gray states in Figure 5.2 must be in the same subgame. The same kind of reasoning can be used to argue that the remaining states must also be grouped together.

Note that the argument above does not mean that there can only ever be one imperfect information subgame. Publicly visible information can be used to partition the game into multiple subgames. In Kuhn poker, if the trunk was the pre-deal state and the 6 initial player one states, we can have two subgames: one where the first player made the left action (corresponding to a "check" action) and one where the player made the right action (corresponding to a "bet" action). Both subgames will include states that differ only in the cards dealt to the player.

The imperfect information subgame can be viewed as an extension of the traditional subgame, in the sense that in perfect information games the imperfect information subgame definition collapses to the traditional definition of a subgame. In perfect information games, the "is in the same information set" relationship has no effect because all information sets contain a single state. Because imperfect information subgames are an extension of the traditional subgame, and they are functionally useful in imperfect information games, I will hereafter use subgame only to refer to an imperfect information subgame.

5.2.2 Subgame Strategy with Fixed Trunk Strategy

The hidden knowledge in imperfect information games, represented with information sets, means subgames have drastically different properties from perfect information subgames. Not only do subgames need to be larger so that they do not break up information sets, there is now a probability distribution across the root states, and that distribution affects the payoffs and consequently affects the optimal subgame strategy. Unlike perfect information games, we can no longer ask questions about subgame strategies independently of the trunk strategy.

Figure 5.3 shows an extensive form game tree for the well known game of rock, paper, scissors. Extensive form games do not have simultaneous moves, so we arbitrarily choose one player to act first, but the second player does not know what action the first player made. As is also shown in Figure 5.3, we can split the game into a trunk containing the initial state, and one subgame containing all three states where the second player acts.

Consider a strategy within the single rock, paper, scissors subgame: a probability distribution over rock, paper, and scissors for a single player two decision. For example, what is the value of player two playing rock? The answer depends on the probability distribution over the states in the information set, which is determined by player one's strategy within the trunk. If player

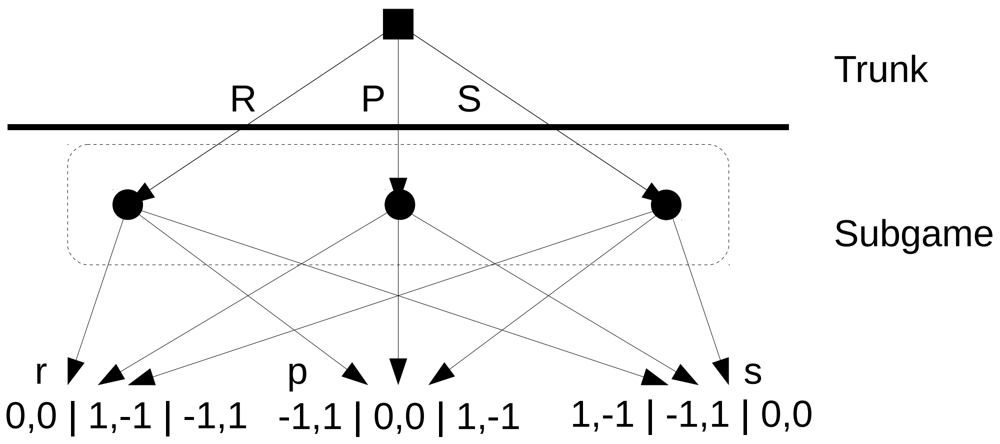


Figure 5.3: Rock Paper Scissors game tree decomposed into trunk and subgame.

one's strategy was to always play paper, the initial distribution of states in the subgame is 0%, 100%, and 0% for the "R", "P", and "S" states respectively, and the value of player two playing rock is -1. If player one's strategy was to always play scissors, the initial distribution of states in the subgame is 0%, 0%, and 100% for "R", "P", and "S" respectively, and the value of player two playing rock is +1.

If we have a fixed trunk strategy, the indeterminacy discussed above is removed. The trunk strategy implies a fixed distribution over the states at the root of the subgame, which turns the subgame into a well defined game[3]. As a well defined game in its own right, the subgame (with fixed trunk strategy) has at least one Nash equilibrium strategy, which can be found with any of the usual game solving algorithms.

5.2.3 Trunk Strategy

At a high level, computing the trunk portion of an equilibrium strategy is similar in perfect and imperfect information games. When we are computing

[3]We must also scale the game by a constant factor to ensure that the probability of the initial states sums to 1. Scaling a game by a constant non-zero factor, however, does not strategically change the game. Note that with N root states, there are then $N-1$ free parameters. With $N=1$, we have 0 free parameters, which is why we do not care about the trunk strategy for perfect information games.

something in the trunk that requires a subgame value, we can use the value of a subgame equilibrium strategy.

In more detail, if we are using an iterative game solving algorithm like CFR, we have a complete trunk strategy profile at each iteration, which means we have well defined subgames that can be solved. Using that same trunk profile, we can compute the counterfactual values of the information sets at the root of the subgame, which is the information needed to update the regrets within the trunk, and generate a new trunk profile for the next iteration. This process describes the CFR-D algorithm [15].

Conceptually, we can see that CFR-D is correct because it is a pair of mutual best responses, and playing a best-response strategy does not increase regret. A formal proof of correctness was published along with the algorithm, and is included in Appendix B. The formal proof notes that we must use a counterfactual best response, so that we are maximising counterfactual value. A counterfactual best response is identical to a standard best response, except where the probability of reaching an information set is 0. In that case, a best-response strategy is unconstrained, but a counterfactual best response must pick a (counterfactual) value maximising action.

We could also consider an optimisation based approach to finding the trunk portion of an equilibrium strategy. Instead of considering $\max_{\mathbf{x}} \min_{\mathbf{y}} \mathbf{x}^{\intercal} A \mathbf{y}$, we could consider $\max_{\mathbf{x}} \min_{\mathbf{y}} \max_{\mathbf{x}_S} \min_{\mathbf{y}_S} \langle \mathbf{x}, \mathbf{x}_S \rangle^{\intercal} A \langle \mathbf{y}, \mathbf{y}_S \rangle$, with the entire inner max-min as the objective function, $\mathbf{x}, \mathbf{y}$ representing the trunk profile, and $\mathbf{x}_S, \mathbf{y}_S$ representing the subgame strategy profile. The objective function is exactly the value of a solution to the subgame, given the trunk profile $\mathbf{x}, \mathbf{y}$. It is interesting to note that because the objective function is not a C^1 continuous function Nemirovski's saddle point algorithm does not apply, and preliminary tests of CFR-D show the expected $\mathcal{O}(1/\sqrt{T})$ behaviour for a non-smooth problem even when using CFR$^+$ instead of CFR to solve the trunk.

In perfect information games, which do not require iteration to find the locally optimal choice, there is no significant cost in computation time if we only generate a trunk strategy. This leads to the common use of algorithms operating on recursively decomposed games. The iterative nature of imperfect information algorithms means that they pay a roughly quadratic computation cost. If we require N iterations for an acceptable approximation error, then we need to solve the subgames N times, and each subgame will require N iterations to solve.

5.2.4 Re-solving a Subgame, in Theory

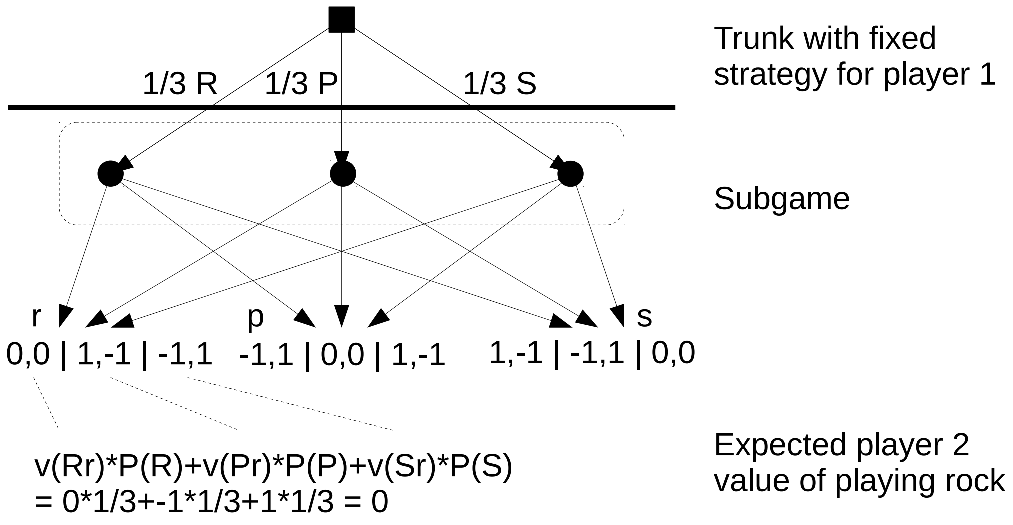
The final tool required for decomposition is some process for re-solving a subgame to get a strategy that is part of an equilibrium strategy when combined with the trunk strategy. For example, CFR-D only produces a trunk strategy, so in order to play a complete game, we need to recover a strategy within subgames. Unlike in perfect information games, re-solving an imperfect information subgame strategy is a different process than solving a subgame, and slightly more complicated. First, like solving a subgame, re-solving a subgame is always relative to a fixed, complete trunk strategy. Second, we need some information about minimum necessary values. Third, the minimum values come from knowledge about an existing strategy in the full game[4].

We can see how the difficulties arise if we return to the rock paper scissors decomposition example of Figure 5.3. In rock paper scissors, the unique equilibrium profile has both players pick rock, paper, and scissors with equal (1/3) probability. Let us say that we have found the trunk portion of this strategy profile, which in this case is the complete strategy for the first player, picking an action uniformly at random. Now, consider finding a profile in the subgame, which in this case is the complete strategy for the second player[5].

If we used the same process used for solving a subgame (Section 5.2.2) we would use the trunk strategy to turn the subgame into a well-defined game and then solve it. Continuing with the rock paper scissors example, this would mean finding the equilibrium profile given that player one picks rock, paper, and scissors uniformly at random, shown in Figure 5.4. In this very small

[4]The use of knowledge about an existing equilibrium is why I chose the name subgame re-solving. We are not creating a subgame equilibrium strategy *ex nihilo*, we are computing a strategy from existing knowledge of an equilibrium strategy.

[5]In general, we would expect both the trunk and subgames to contain strategic information for both players. Rock paper scissors, with a single simultaneous action, has insufficient structure for interesting decomposition.



Trunk with fixed strategy for player 1
1/3 R
1/3 P
1/3 S
Subgame
r
p
s
0,0 | 1,-1 | -1,1
-1,1 | 0,0 | 1,-1
1,-1 | -1,1 | 0,0
v(Rr)*P(R)+v(Pr)*P(P)+v(Sr)*P(S)
= 0*1/3+-1*1/3+1*1/3 = 0
Expected player 2 value of playing rock

Figure 5.4: Incorrect Re-solving of Rock Paper Scissors Subgame

example, player one has no actions in the subgame, and the problem collapses down to finding a player two strategy which maximises utility against the fixed player one strategy. One such strategy is to always play rock: it has an expected value of 0 against uniform random rock, paper, scissors (as does any strategy.) We have gone wrong, however, because player two always playing rock is not part of any equilibrium profile in the full game. Always rock maximises value against a fixed opponent trunk strategy, but does not maximise value against an arbitrary opponent trunk strategy.

To get a subgame strategy that is part of an equilibrium strategy in the full game, we need to meet or exceed some certain level of performance against an arbitrary opponent. If our new strategy does as well as an actual equilibrium strategy against a worst-case opponent, our new strategy must also be part of an equilibrium strategy in the full game. We can get this guarantee by ensuring the opponent's counterfactual values at the root of a subgame are no higher than opponent counterfactual values computed from an equilibrium profile that uses our current trunk strategy. This is because, for any opponent strategy, the subgame's contribution to their expected value will be a product of the counterfactual values at the root of the subgame and the opponent's probabilities of reaching the subgame.

To re-summarise the fundamental point, given our fixed trunk strategy, the opponent counterfactual values at the root of a subgame provide a summary of the opponent's best-case performance, for any trunk strategy they could choose to play. If we compute opponent best-response values from an existing subgame strategy, whether or not it is part of an equilibrium, we can use the values as an upper bound constraint to find ourselves a new subgame strategy that is no more exploitable than the original strategy.

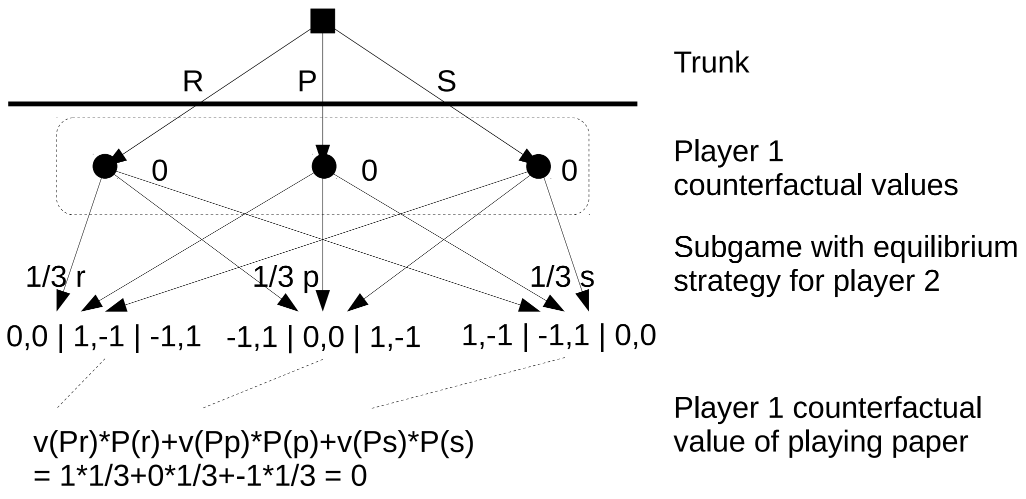

Trunk
R
P
S
0
0
0
Player 1 counterfactual values
Subgame with equilibrium strategy for player 2
1/3 r
1/3 p
1/3 s
0,0 | 1,-1 | -1,1
-1,1 | 0,0 | 1,-1
1,-1 | -1,1 | 0,0
Player 1 counterfactual value of playing paper
v(Pr)*P(r)+v(Pp)*P(p)+v(Ps)*P(s)
= 1*1/3+0*1/3+-1*1/3 = 0

Figure 5.5: Computing Opponent Counterfactual Values at Subgame Root

Examining opponent counterfactual value constraints in the context of the ongoing rock, paper, scissors example, let us say we are player two. As part of the process of computing the trunk portion of an equilibrium profile, we must have also computed the subgame portion of the profile (even if only implicitly, as in CFR-D) where player 2 also acts uniformly randomly, shown in Figure 5.5. If we compute the opponent (player one) best-response counterfactual values for the subgame (or implicitly track them, as in CFR-D) we see that our opponent gets a value of 0 after all three actions[6].

Figure 5.6 revisits the situation where we incorrectly re-solved the subgame, and chose a player two strategy that always plays rock. Using 0 as a constraint rules out always rock, because player one choosing paper has a value of 1

[6]While we, as player two, cannot distinguish the three states at the root of the subgame, the three states are all distinct to our opponent, player one.

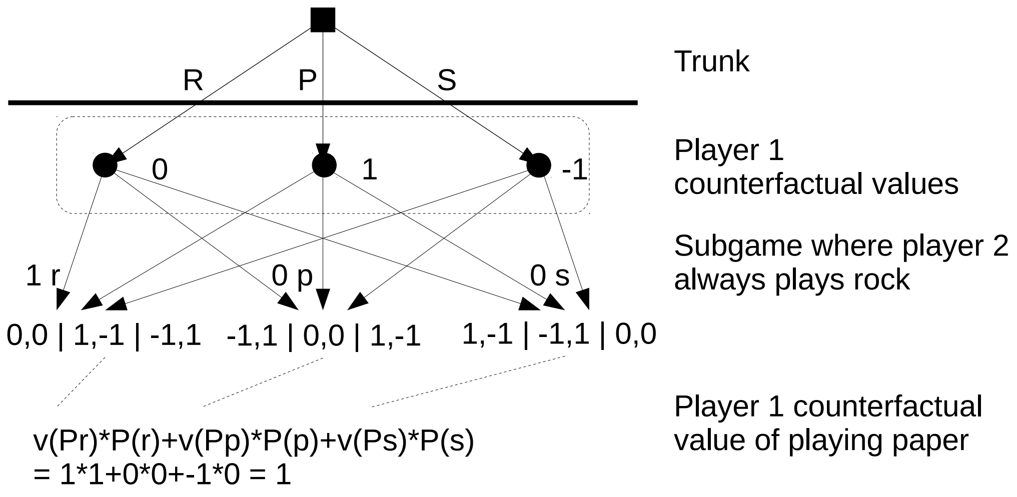

Trunk
R
P
S
Player 1 counterfactual values
0
1
-1
Subgame where player 2 always plays rock
1 r
0 p
0 s
0,0 | 1,-1 | -1,1
-1,1 | 0,0 | 1,-1
1,-1 | -1,1 | 0,0
Player 1 counterfactual value of playing paper
v(Pr)*P(r)+v(Pp)*P(p)+v(Ps)*P(s)
= 1*1+0*0+-1*0 = 1

Figure 5.6: Opponent Counterfactual Values for Always Rock

against always rock, which exceeds our constraint of 0 from the equilibrium in Figure 5.5. To re-solve the game, we must find a strategy where player one gets an expected value of no more than 0 for every action, which only occurs with the original player two equilibrium strategy of uniform random actions. Note that while we ended up with an identical strategy after re-solving the rock paper scissors subgame, more generally our re-solved subgame strategy is only guaranteed to be at least as good as the original strategy against an opponent best response, not to be identical to the original strategy.

5.2.5 Re-solving a Subgame with a Gadget Game

Section 5.2.4 described how, given a trunk and subgame strategy for one player 1, we can summarise the subgame strategy by constraints on the opponent's best-response counterfactual values at the root of a subgame. We can keep the constraint values and discard the subgame strategy, because for any new subgame strategy with constraint values that do not exceed the original constraint values, the combination of the trunk strategy and the new strategy is no more exploitable than the trunk and original strategy. This section describes how we can build a special re-solving game from the constraint values, such that player 1's equilibrium strategy satisfies the constraints in the original subgame.

First, we must more carefully define what we mean by opponent counterfactual values at the root of the subgame. Let us say we are player 1 and our opponent is player 2. Because player 1 might be acting in the root states R and information sets partition the states according to the player that is acting, we need something like "opponent information sets" to describe the root from player 2's point of view. We partition the states R according to the sequence of player 2 information sets and player 2 actions to reach the states, giving us opponent "information sets" $\mathcal{I}_R$. That is, as in Section 2.3.2, a state is a history of actions, which induces a sequence of tuples $< I_i, a_i >$, from which we can take the subsequence of tuples where $p(I_i) = 2$. States $h, j \in R$ are in the same $I \in \mathcal{I}_R$ if and only if h and j induce the same subsequence.

For all "opponent information sets" $I \in \mathcal{I}_R$, there is an equivalence between the expected value in the re-solving game, and the counterfactual value in the original game. The counterfactual value $v_2^{\sigma_1}(I) = \sum_{h \in I} v_2^{\sigma_1}(h) = \sum_{h \in I} \pi_{-2}^{\sigma_1^T}(h) u_2^{\sigma^S}(h)$ where σ_1^T is the trunk strategy for player 1 and σ^S is the subgame strategy profile. The opponent's best-response counterfactual value[7] $\text{vbr}_2^{\sigma_1^T \sigma_1^S}(I) = \max_{\sigma_2^S} v_2^{\langle \sigma_1^T \sigma_1^S, \sigma_2^S \rangle}(I)$ can be used as the constraint values that are part of the input to a re-solving problem. That is, given constraint values c indexed by opponent information set I, we want to enforce $\text{vbr}_2^{\sigma_1^T \sigma_1^S}(I) \le c_I$, which is equivalent to enforcing $u_2^{\sigma_1^S}(I) \le c_I$ in the re-solving game.

Figure 5.7 shows the re-solving game, which uses a gadget at the beginning of the game to ensure that constraint values are satisfied. Conceptually, the opponent has a choice for every information set whether to terminate and achieve the constraint value, or follow through and play within the subgame.

We start by duplicating the states R at the root of the subgame, giving us new states $\tilde{R} = \{\tilde{r} | r \in R\}$. For all $\tilde{r} \in \tilde{R}$ there is a terminal action T and a non-terminal F that reaches state r. $P(\tilde{r}) = 2$, regardless of what player was acting in state r. $\tilde{R}$ is partitioned according to the opponent information partition $\mathcal{I}_R$, giving us information sets $\mathcal{I}_{\tilde{R}}$.

In order to make it a proper game, we need an initial chance decision that

[7]Note that a counterfactual best response in a subgame is subtly different than a best response within the full game. A best response within the full game considers $\pi_2(I)$ and thus will not maximise counterfactual value for states that opponent 2 does not play to.

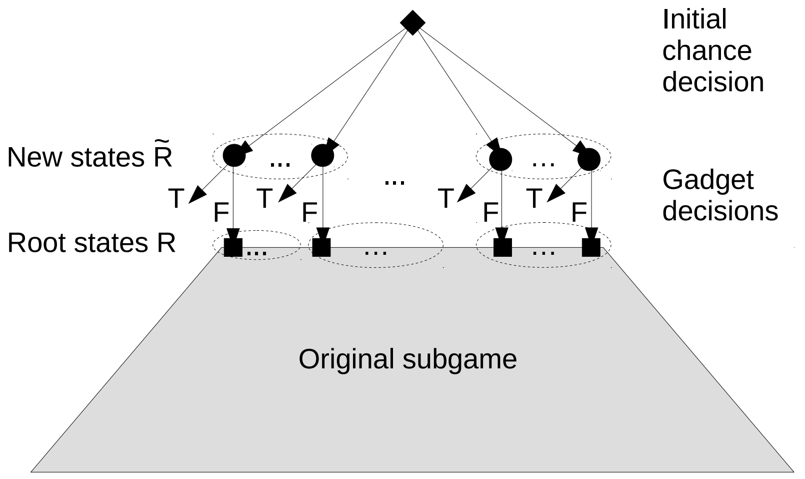


Figure 5.7: Construction of Re-Solving Gadget Game

reaches all states $\tilde{r} \in \tilde{R}$ with probability $\pi_{-2}^{\sigma^T}(r)/k$, where $k = \sum_{r \in R} \pi_{-2}^{\sigma^T}(r)$ is a normalising constant so the probabilities sum to 1. To keep the theoretically useful property that expected values in the re-solving game correspond to counterfactual values in the subgame (*i.e.,* $\tilde{u}_2(h) = v_2(h)$) then we must also multiply all leaf values by k so that $\tilde{u}(z) = ku(z)$.

Finally, we need to assign values to the T actions in the new gadget decisions, using the vector of constraint values c supplied as an argument to the re-solving problem. For every information set $I \in \mathcal{I}_{\tilde{R}}$ and for all $r \in I$, we set $u_2(r \cdot T) = kc_I / \sum_{r' \in I} \pi_{-2}(r')$. This choice means that, considering the initial chance event and the sum across $r \in I$, the expected value $u_2(I \cdot T)$ for choosing T at information set I is equal to the constraint value.

Informally, we maximise our value in the re-solving game by minimising opponent counterfactual values in the original game. The gadget allows the opponent to opt out of playing into the subgame with any of their information sets, so that we have no incentive to minimise one opponent value at the expense of increasing another opponent value beyond the original constraints. If an opponent best response to some existing strategy is used to compute the constraints, we know that it must be possible to have the values be at least as low as the constraints, so our re-solving equilibrium strategy must be at least as good as the original strategy. A formal proof that the re-solving game can be used to generate a subgame strategy that does not increase exploitability can be found in Appendix B.

5.3 Solving Games Using Decomposition

We now have all the pieces necessary to solve and play games using decomposition. Given a trunk and imperfect information subgames, we can solve the game to get a trunk strategy using CFR-D, which requires us to repeatedly solve subgames (with any algorithm, including CFR-D). When needed, we can recover a strategy in the subgames by re-solving.

For example, let us say we wanted to play the game of HULHE and we did not have access to the tens of terabytes used to generate the Cepheus solution, but did have access to a machine with 16GiB of memory and wanted to use CFR-D. We could consider splitting the game after the second round of betting, just before the fourth public card is dealt. This would give us a trunk with $3.8 * 10^8$ betting sequences, and each of the 142 155 subgames would have $4.6 * 10^8$ betting sequences. This arrangement is shown in Figure 5.8.

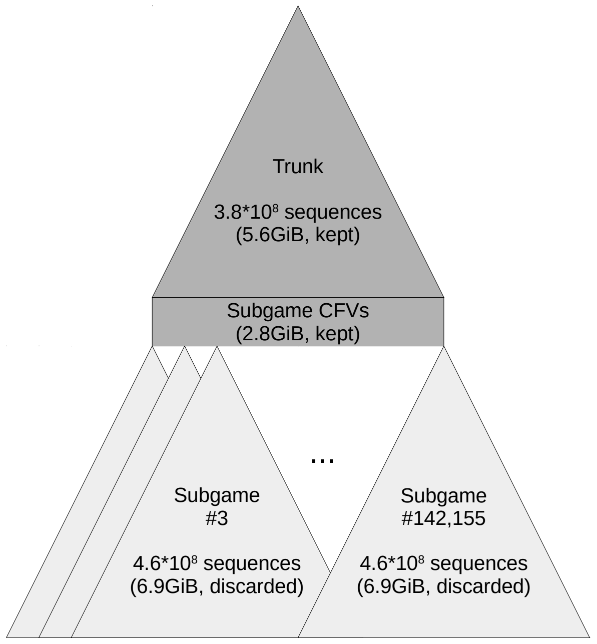


Figure 5.8: Using CFR-D on HULHE

The CFR-D computation will generate the trunk portion of an equilibrium profile, covering the top dark-shaded triangle. In order to later re-solve a subgame, CFR-D must also store opponent counterfactual values for each subgame, shown as the dark-shaded rectangle below the trunk. Because we might need to act as either player while playing, we must store subgame counterfactual values for both players. During the CFR-D computation, there will be repeated calls to a game solver to solve a subgame. However, because we only need the counterfactual values of the subgame profile, the space required by the subgame solver can be freed up after the subgame computation is finished. At 8 bytes per value, we only require a bit more than 15GiB.

When it comes time to play the game, what do we do? While the current state remains within the trunk, everything is easy: we just look up the action distribution from our stored trunk strategy. Once we hit a subgame, we must now look up the stored opponent counterfactual values for the appropriate subgame and player, and re-solve the subgame as described in Section 5.2.5. We then use that re-solved subgame strategy until the end of the game, with no further computation needed. Note that the use of re-solving means that we now have a (potentially significant!) computation cost to playing a game.

There is a cost to the reduction in space, from 262 TB even with 4 byte values [9] down to less than 16 GiB with 8 byte values. Because CFR-D requires a solution to the subgames at every iteration, and the subgames comprise the majority of the game, the computation cost is roughly squared. If we use CFR or CFR$^+$ to solve the subgames, and use N iterations both for the CFR-D trunk computation and for solving subgames, we will update values for subgame information sets N^2 times. Additionally, as mentioned in Section 5.2.3, CFR-D modified to use CFR$^+$ in the trunk does not have the empirically observed $\mathcal{O}(1/T)$ behavior of CFR$^+$ by itself.

5.3.1 Algorithm

Algorithm 6 gives a simple version of CFR-D. The input T describes the number of trunk iterations. `Solve` is the algorithm used to solve subgames, and could be CFR, an LP-based method, or CFR-D itself (with its own argument of a solver for subgames). Finally, CFR-D requires a description of how the game game is divided into a trunk and subgames. After running, CFR-D returns the average strategy profile in the trunk, as well as the average values at the root of each subgame. The returned subgame values can be used as constraint values for re-solving.

At each iteration, after computing the current trunk strategy, we solve each subgame, one at a time. The trunk strategy defines a probability distribution over states at the root of a subgame, making it a well defined game that we can solve. Generating this new game is described in `MakeSubgameGame`. After solving the subgame, we find the best response strategy profile, and compute the counterfactual values at the root of the game for each player when playing the solution against the best response. This final best response step guards against irrational (non best-response) play in the solution at any information sets not reached by the trunk, as described in Section 5.2.3.

After computing the subgame values we have all the information needed to compute counterfactual values in the trunk. As in the recursive version of CFR, we make use of the fact that the counterfactual values can be decomposed into other counterfactual values. Say that $C(I \cdot a)$ is the set of child information sets for information set I after making action a, where the same player is acting[8]. Then $v_p(I,a) = \sum_{J \in C(I)} v_p(J)$, for $p = P(I)$. With $v_p(I)$ computed for information sets at the root of every subgame, we can compute $v_p(I,a)$ for every information set within the trunk.

After computing subgame values, CFR-D updates $\bar{v}_{\mathcal{S}}$, the average counterfactual value for every information set at the root of each subgame, averaged across all time steps. Finally the iteration finishes as in CFR, by computing values in the trunk, and using them to update regrets in the trunk.

5.3.2 Comparison to Double Oracle Methods

Double oracle methods [63] also let us solve a reduced problem, using a best response as a computable oracle. We start by solving a reduced problem

[8]As in Section 2.4.3, $C(I \cdot a) = \{J \in \mathcal{I}_{p(I)} | I \cdot a \sqsubset J, \nexists K \sqsubset J \text{ s.t. } I \cdot a \sqsubset K\}$, where $I \sqsubset J$ if and only if $\exists h \in I, j \in J$ such that $h \sqsubset j$.

```
input  : Game $\mathcal{G}$, Number of iterations $T$, game solving algorithm Solve,
         trunk information sets $\mathcal{T}$, set of subgames $\mathcal{S}$
output: Trunk strategy of $\epsilon$-Nash approximation $\bar{\sigma}$,
         average player values at root of subgames
foreach $I \in \mathcal{T}$ do
    foreach $a \in A(I)$ do
        $R(I,a) \leftarrow 0$
        $\bar{\sigma}(I,a) \leftarrow 0$
foreach $S \in \mathcal{S}$ do
    foreach $p \in \{p_1, p_2\}$, information set $I_p$ at root of $S$ do
        $\bar{v}_{\mathcal{S}}(I_p) \leftarrow 0$
for $t \leftarrow 1$ to $T$ do
    $\sigma \leftarrow$RegretToStrategy($\mathcal{T}$,$R$)
    UpdateAverage($\mathcal{T}$,$\sigma$,1,$\bar{\sigma}$)
    $v_{\mathcal{S}} \leftarrow$ SubgameValues($\mathcal{S}$,$\sigma$,Solve)
    $\bar{v}_{\mathcal{S}} \leftarrow (t-1)/t\bar{v}_{\mathcal{S}} + 1/tv_{\mathcal{S}}$
    $v_{\mathcal{T}} \leftarrow$ StrategyToValues($\mathcal{T}$,$\sigma$)
    UpdateRegrets($\mathcal{T}$,$\sigma$,$v$,$R$)
NormaliseAverage($\mathcal{T}$,$\bar{\sigma}$)
return $\bar{\sigma}$, $\bar{v}_{\mathcal{S}}$

function SubgameValues($\mathcal{S}$,$\sigma$,Solve):
    foreach $S \in \mathcal{S}$ do
        foreach state $h$ at root of $S$ do
            $probs[h] \leftarrow \pi^{\sigma}(h)$
        $Game \leftarrow$ MakeSubgameGame ($\mathcal{G}$, S, probs)
        $\sigma_S \leftarrow$ Solve ($Game$)
        $\sigma_{BR} = \langle br_{p_1}(\sigma_S), br_{p_2}(\sigma_S) \rangle$
        foreach $p \in \{p_1, p_2\}$, information set $I_p$ at root of $S$ do
            $v(I_p) \leftarrow v_p^{\langle \sigma_{BR,p}, \sigma_{S,-p} \rangle}(I_p)$
return $v$
function MakeSubgameGame($\mathcal{G}$, $S$, $probs[\cdot]$):
    input  : Game, subgame, root probabilities
    output: New game containing subgame states
    $k \leftarrow \sum_{h \in S} probs[h]$
    $H_G \leftarrow \{h \in S\} \bigcup \varnothing$
    $Z_G \leftarrow Z_{\mathcal{G}} \bigcap H_G$
    $p_G(h) \leftarrow p_{\mathcal{G}}(h)$ if $h \in S$ else $c$
    $u_G(h) \leftarrow k u_{\mathcal{G}}(h)$
    $\sigma_{c,G}(h,a) \leftarrow \sigma_{c,\mathcal{G}}(h,a)$ if $h \in S$ else $probs[a]/k$
    return $\langle H_G, Z_G, P_{\mathcal{G}}, p_G, u_G, \mathcal{I}_{\mathcal{G}}, \sigma_{c,G} \rangle$
```

Algorithm 6: CFR-D

using some subset of the possible strategies for each player. Then, for each player, we look at a best response to the strategy within the full game. If the player's best response improves their value, it is added to the reduced problem. Because it is using best responses, we get information about the quality of the solution, and can stop when the solution to the reduced problem is an adequate approximation of an equilibrium in the full game.

For some problems, incrementally adding strategies can be very effective. In some security games, one player must find a path, and the other player is selecting locations [7, 24, 41, 42]. This class of games can quickly grow to have an astronomically large number of information sets: the number of possible paths grows exponentially with the maximum path length. An acceptable solution, however, might be found using only hundreds of strategies.

Double oracle methods can reduce space requirements, and frequently can reduce computation costs as well by working with a smaller problem at each iteration. Their effectiveness, however, depends on there being a small support for the equilibrium, so that we only need a small set of strategies. In contrast, CFR-D and decomposition offers a guaranteed, controlled reduction in space, but at the cost of increased computation time.

5.4 Re-solving Exploitable Strategies

The re-solving technique of Section 5.2.5 was discussed in the context of solving a game, but it also has other uses. Given a trunk strategy and opponent best-response counterfactual values to our subgame strategy, re-solving lets us generate a new subgame strategy that is no more exploitable than the original subgame strategy, when combined with the trunk strategy. We can use that guarantee to save space, as in Section 5.3 where we immediately discard subgame strategies, or we could also use it to try improve an existing strategy.

Improving a strategy works as follows. The opponent best-response counterfactual values are a constraint on the subgame strategies we can consider, in the form of an upper bound on the performance of an opponent best response against our strategy. If our strategy cannot be improved, these bounds will be tight, and there is no strategy for us that lowers the value for one opponent information set without increasing the value at another information set. However, if our strategy can be improved, these bounds are loose. If we re-solve the subgame using those bounds and the gadget of Section 5.2.5, there is no guarantee that we will decrease opponent values below the bounds and thus improve our strategy, but we are likely to do so as the bounds are loose.

We could use re-solving to reduce exploitability if we know, or strongly suspect, that our current strategy is exploitable in some portion of the game. As a motivating example, consider the use of game abstraction, as demonstrated in attempts to approximate equilibrium strategies in poker games [79, 4, 33]. Instead of directly solving the game, we first construct a small version of the game, an abstraction which tries to capture the strategically important elements of the original game. We then solve the small abstract game, and finally translate that strategy back into the original game. With few exceptions [53], there are no bounds on the exploitability of the translated solution. We would expect that in any subgame we looked at, the translated strategy would be a suitable candidate for re-solving to reduce exploitability.

In particular, many abstract games for poker games are constructed by grouping together card combinations into abstract buckets: instead of a player knowing that they have $3\heartsuit K\spadesuit$ with $7\heartsuit 8\diamondsuit A\heartsuit 6\clubsuit 5\clubsuit$ as public cards, the player might only know their cards are in bucket 2732 out of 10 000. The abstract strategy will be for all of the situations grouped into that bucket, not just that single deal of cards. If we ran across that situation in a real game, we might want to use the knowledge of the exact cards.

The story of using exact knowledge of the real game state to fix problems with abstraction is compelling enough that it was done even without theoretically sound re-solving techniques. Unpublished versions of the PsOpti agent [4] used the solving process of Section 5.2.2 in the final river round, starting with the fixed probability distributions for both players from the strategy profile in earlier rounds. With the technology of the time, the abstract games needed to be very small, and results were slightly negative. Independently discovered by

Ganzfried *et al.*, a similar unsafe subgame solving technique that eliminates dominated strategies was used with positive results [28].

5.4.1 Experimental Results

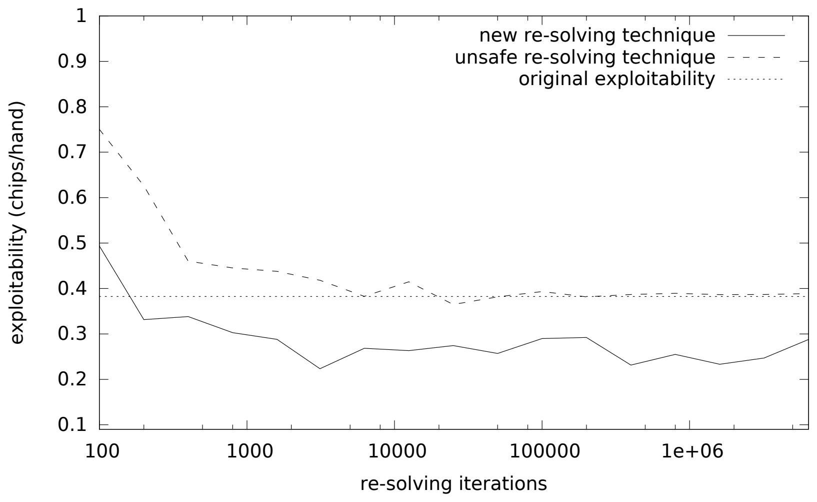


Figure 5.9: Exploitability of re-solved abstract strategy in Leduc hold'em

Figure 5.9, originally published at AAAI [15], looks at the exploitability after re-solving a strategy generated from an abstraction of Leduc hold'em. It looks at both the safe re-solving method in Section 5.2.5 that has a theoretical guarantee, as well as the unsafe method of Ganzfried *et al.* Both methods used CFR to solve the subgame re-solving games, and the exploitability is plotted against the numbers of iterations used to solve the games. The original strategy used was exploitable for 0.382 chips/game.

We can see a strong improvement in the exploitability using the safe re-solving method. Within 200 CFR iterations, which is only sufficient to produce a poor approximation of an equilibrium profile, we already reduce the exploitability to 0.33 chips/game. After 2000 iterations, the exploitability varies between 0.23 to 0.29 chips/game, or a reduction of 24%-40%. The value does not converge, because the equilibrium strategies in the re-solving gadget game only provides a guarantee that the exploitability does not increase, with no preference for strategies which maximise the decrease in exploitability. In contrast, after 6250 iterations the unsafe method reaches and maintains an exploitability of 0.39 chips/game. The apparent convergence to the original exploitability is a coincidence: in other situations the unsafe strategy could be less exploitable, or significantly more exploitable.

Experimental Design

The original strategy used for the Leduc hold'em re-solving experiment was generated using an abstract game of the style discussed above in Section 5.4. In the abstract game, players cannot distinguish between a Queen or a King on the board if they hold a Jack, or between a Jack or a Queen on the board if they hold a King. That is, they know their card perfectly on the first round, but can only distinguish JJ, JQ/JK, QJ, QQ, QK, KJ/KQ, and KK on the second round.

This abstract game still lets players exactly compute the expected outcome of their hand against a uniform random opponent hand (commonly known as "hand strength"), so we might consider it to have captured much of the complexity in the original game. The players, however, do lose information about value against a non-uniform opponent hand: if I had the Jack, knowing whether the board was the Queen or King would influence my belief about the opponent holding a Queen, which changes how successful I think a bluff should be. Because of this missing information, the equilibrium strategy from the abstract game is exploitable for 0.382 chips/game in the full game[9].

One natural choice for a trunk in the game of Leduc hold'em is the first round of betting. If we split the game after the public cards are dealt, we have 30 subgames, corresponding to the six possible public cards and the five betting sequences where neither player ends the game early by folding in the first round. Then, with a strategy, trunk and subgames in hand, we compute two vectors for each subgame. First, we need our probability of reaching the subgame for each information set (*i.e.,* $\pi_{\text{us}}(I)$), computed using a product of action probabilities in the trunk. Second, we need our opponent's best-response counterfactual values for each opponent information set, computed

[9]Note that the exploitability is for one particular equilibrium. There are multiple abstract game equilibrium profiles, with different exploitability in the full game.

with the standard recursive best-response algorithm, which naturally produces counterfactual values. We do this computation twice, so we have this information from both players' point of view.

With the probabilities and counterfactual values for each subgame, we can now use the method of Section 5.2.5 to produce a new strategy. Note that we must run this computation twice for each subgame to get a strategy profile, once for each player, as the re-solving computation only has a guarantee on the strategy for one player. Combining the new subgame strategy profiles with the trunk strategy profile, we get a new strategy profile which is guaranteed to be no more exploitable than the original strategy profile.

We can also use the unsafe method of Ganzfried *et al.* [28]. This method only requires the probabilities of reaching the subgame, as in the computation of Section 5.2.2. To improve the quality of the solution, it also forces the probability of playing any dominated strategy to 0. The unsafe method only needs to be run once per subgame, as we can use the strategy profile, rather than just the strategy for a single player. Combining the new subgame profiles with the trunk, we get a third strategy profile which is hopefully better than the original, but could potentially be more exploitable.

One on One Performance

Because there can be a disconnect between exploitability and the one-on-one performance [47], we might also be interested in the effect of re-solving on the expected performance against other agents.

Figure 5.10 looks at the expected performance of the re-solved strategies against the original strategy (*i.e.,* the strategy used in the re-solving process.) The unsafe re-solving method quickly converges to an expected value of 0.032 chips/game when playing against the original strategy. With somewhat more computation, the safe re-solving technique achieves a value of 0.028 chips/game against the original strategy.

The unsafe re-solving method outperforms the safe re-solving method by around 0.004 chips/game in this experiment. However, playing against the original strategy is the ideal situation for the unsafe re-solving method, be-

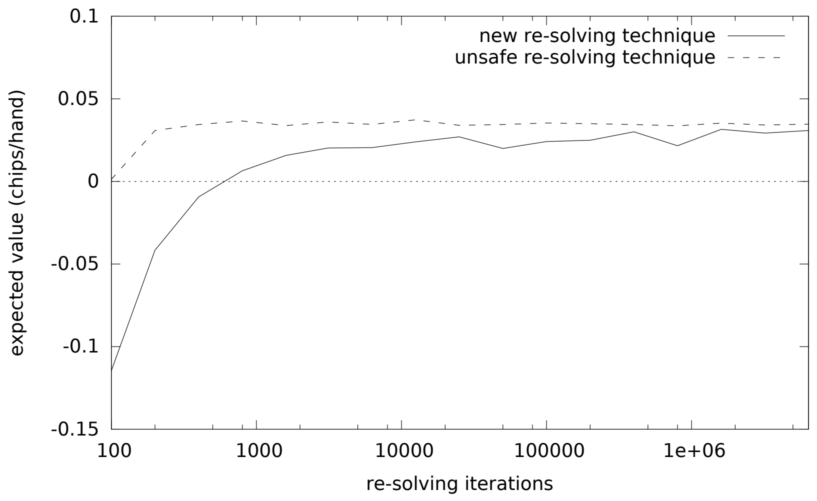


Figure 5.10: Performance of re-solved abstract strategy in Leduc hold'em against the original strategy

cause it assumes that the opponent trunk strategy does not change, and the assumption is true when playing against the original strategy.

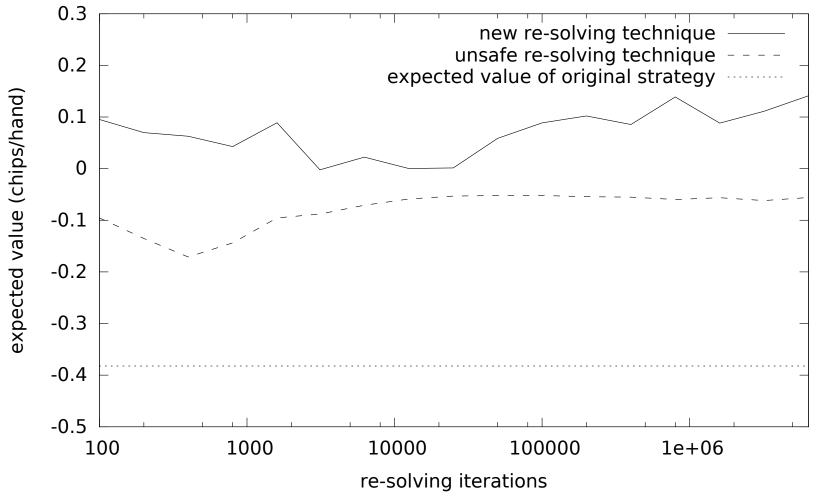


Figure 5.11: Performance of re-solved abstract strategy in Leduc hold'em against a best response to the original strategy

Figure 5.11 looks at the expected performance of the re-solved strategies against a strategy which is a best response to the original strategy. Unlike when we played against the original strategy, the trunk strategy has now changed and the unsafe re-solving method is making false assumptions. Safe re-solving now outperforms unsafe re-solving by around 0.2 chips/game.

5.4.2 Improving Re-solving

There are a number of improvements to the re-solving process, since the original publication. I explored a different source of values for the re-solving constraints, and tried mixing opponent probabilities into the re-solving process. This is previously unpublished work. In related work, Moravčík *et al.* published a different re-solving gadget that combines the exploitability guarantee with an objective function [66].

Re-solving with Self-play Values

All of the re-solving theory in the discussion above uses the opponent best-response counterfactual values $\mathrm{vbr}^{\sigma}(I)$ from an existing strategy. A best response, however, is clearly pessimistic and the self-play values $v_2^{\sigma}(I)$ of abstract strategies can be a better source of constraints. For example, if we re-run the experiment of Section 5.4.1 using self-play values, we get an exploitability of 0.143 chips/game instead of 0.288 chips/game, a reduction of 50%. The remainder of this section examines the reasons why self-play values might work well in practice.

Consider how the constraint vector c supplied as part of the re-solving process affects the exploitability. For simplicity, let us assume we are some fixed player 1 with strategy σ_1, our opponent is player 2, we are considering a single subgame S, and c is the vector of constraint values for σ_1^S (*i.e.,* each c_I is the opponent player 2 best-response counterfactual value $\mathrm{vbr}_2^{\sigma_S}(I)$.)

Say that we re-solve S using constraints c, and get some new subgame strategy $\sigma_1'^S$ with constraint values c'. The expected (not counterfactual) value of S for player 2 using any best response σ_2^* is $\pi_2^{\sigma^*}(S) \cdot c'$. So, if the values in c' are lower than c, we have potentially decreased player 1's exploitability. Note that the improvement in exploitability is not guaranteed because a best-response opponent might already choose not to play into the subgame with that information set, so that reducing the value even further has no effect.

As long as c represents feasible constraints, the new values c' will be as good as c. That is, if there is some $\tilde{\sigma}_S$ with constraint values $\tilde{c}$ such that $c \geq \tilde{c}$

($i.e.$, $c_I \geq \tilde{c}_I \forall I$), then the re-solving theorem guarantees that $c' \leq c$. If the constraints are not feasible, the re-solving gadget game is still well defined, so that we still get some new $\sigma_1'^S$, but there is no longer a guarantee that $c' \leq c$.

There are, then, two broad classes of "error" in c if we are trying to minimise our exploitability: overestimation and underestimation. If the values in c are too high, so that we are too pessimistic about what values the opponent can achieve, we may get a subgame strategy which is more exploitable than it needs to be. On the other hand, if the values in c are overly optimistic and too low, we may also get an unnecessarily exploitable strategy. Fortunately, the increase in exploitability is bounded by the amount of underestimation.

Theorem 11 *Assume we have a strategy σ_1 for player 1 with opponent 2, and some subgame S. Let c be re-solving constraints such that $c_I \leq v_2^{br(\sigma^S)}(I)$ and $\epsilon + \sum_I c_I = \sum_I v_2^{br(\sigma^S)}(I)$ for some $\epsilon > 0$, and σ'^S be the strategy from re-solving S using the gadget game of Section 5.2.5 with constraints c. Then the exploitability of σ_1 combined with σ'^S is no more than ϵ greater than σ_1.*

Proof. First, note that because $c \leq v_2^{\mathrm{br}(\sigma^S)}$, if player 1 plays σ_1^S then player 2 will achieve at most $\sum_I v_2^{\mathrm{br}(\sigma^S)}(I)$. Therefore we get a bound on the player 2 value of the gadget game,

$$\sum_I \max(c_I, v_2^{\mathrm{br}(\sigma'^S)}(I)) \leq \sum_I v_2^{\mathrm{br}(\sigma^S)}(I) = \epsilon + \sum_I c_I$$

Subtracting c_I from each side inside the sum

$$\sum_I \max(0, v_2^{\mathrm{br}(\sigma'^S)}(I) - c_I) \leq \epsilon$$

Noting again that $c \leq v_2^{\mathrm{br}(\sigma^S)}$

$$\sum_I \max(0, v_2^{\mathrm{br}(\sigma'^S)}(I) - v_2^{\mathrm{br}(\sigma^S)}(I)) \leq \sum_I \max(0, v_2^{\mathrm{br}(\sigma'^S)}(I) - c_I) \leq \epsilon$$

If the player 2 best response to σ'^S only played into information sets where $v_2^{\mathrm{br}(\sigma'^S)}(I) - v_2^{\mathrm{br}(\sigma^S)}(I) > 0$, they can increase their value by at most ϵ. ■

Finally, we can consider how best-response and self-play values differ, when starting with an exploitable strategy. If we are exploitable, we must have a less exploitable strategy (*e.g.,* the 0 exploitability Nash equilibrium is guaranteed to exist), so the opponent best-response counterfactual values must be overestimates. The self-play values are likely to be lower, reducing the amount of overestimation, but some values may be infeasible underestimates. In practice, strategies from abstract games appear to have self-play values which are close to equilibrium values, while the best-response values can be very high. The trade-off then favours using self-play values, as seen in the Leduc results at the beginning of this section.

Conversely, if we are starting with a strategy profile where the exploitability is already low, the self-play values must be close to best-response values, and Theorem 11 tells us there is at most a small penalty to using self-play values rather than best-response values.

Using Opponent Probabilities in Re-solving

The safe re-solving method does not increase exploitability, but does not have any additional objective that necessarily decreases exploitability. We can adapt an idea of the robust counter-strategy method of Johanson *et al.* [48] to provide an alternate objective. Johanson *et al.* use a coin-flip gadget to make a trade off between exploitability and exploitation of a fixed strategy. At the beginning of the game, there is a single binary chance event, with the outcome unknown to the player of interest. With probability p, we play a game where the opponent plays the fixed strategy, and with probability $1 - p$ we play the normal game where the opponent is free to react to us.

The idea is to use an estimate of opponent probabilities to construct an additional objective. In the re-solving game as presented above, once the opponent stops playing into some information set $I \in \mathcal{I}_R$ (*i.e.,* by selecting T with probability 1 in the the re-solving gadget at the previous information set $\tilde{I} \in \mathcal{I}_{\tilde{R}}$), we lose incentive to continue minimising the opponent counterfactual value $v_2^\sigma(I)$. This happens because $\pi_2^\sigma(h) = 0$ for all states h that can be reached from I, so $v_1^\sigma(h) = 0$. If we forced player 2 to play into I, the values could be non-zero and we would still have incentive to decrease $v_2^\sigma(I)$.

The additional objective is built by using a fixed opponent policy at all

$\tilde{I} \in \mathcal{I}_{\tilde{R}}$, setting $\sigma_2(\tilde{I}, F) = q_I$. Like c, q is a vector indexed by opponent information sets, but q specifies an estimate of $\pi_2^\sigma(I)$. With probability p we play the game using this new objective, and with probability $1 - p$ we play the unmodified re-solving game. As long as q_I is a decent estimate of $\pi_2^\sigma(I)$ for player 2's actual strategy $\pi_2^\sigma(I)$, we will do a decent job at estimating how important it is to minimise the value of each information set.

To use the coin-flip idea in the re-solving game, we add the coin-flip decision before the re-solving gadget, with the result being hidden from the re-solving player. With probability p we use the opponent trunk probabilities $\pi_2^{\sigma^T}(I)$ when playing into the game, and with probability $1 - p$ the opponent plays into the game through the re-solving gadget. Note that because the opponent's counterfactual values in the main game are unaffected by any gadget decision, we do not need separate opponent information sets for the coin-flip decision.

Using the opponent ranges appears to have a strong, mostly positive effect, and is robust with respect to the choice of p. Even better, using the opponent ranges often decreases the exploitability rather than increasing it, despite the trade-off the coin-flip gadget is making between two different objectives. For an example, consider re-solving the same abstract strategy used throughout Section 5.4.1 with the coin-flip gadget, using both best-response and self-play values. In these experiments, $p = 0$ corresponds to re-solving without the coin-flip gadget, and $p = 1$ corresponds to the unsafe re-solving method.

Figure 5.12 looks at the expected performance against the original strategy. As we might expect, using the opponent probabilities increases our value against that opponent, with the best results coming from large values of p. If we only look at the results when using best-response values for re-solving constraints (the solid line), even $p = 0.01$ offers substantial improvement, and the outcome does not vary much across different values of p until we hit $p = 0$ or $p = 1$. We get one negative result when using self-play values for constraints (dashed line), where the results without the coin-flip ($p = 0$) are better than using the coin-flip until $p \geq 0.8$.

Figure 5.13 looks at the exploitability of the re-solved strategy. The main result is that the exploitability is lower for any value of $p \in [0.01, 0.9]$, and is still lower at $p = 0.99$ when using best-response values as re-solving constraints. As with the one-on-one performance, there is a sharp transition at $p = 0$ and $p = 1$, with both extremes being more exploitable than any other choice of p. Although the effect is small when using self-play values as re-solving constraints, with $p = 0.8$ we both reduced the exploitability, and increased the one-on-one performance.

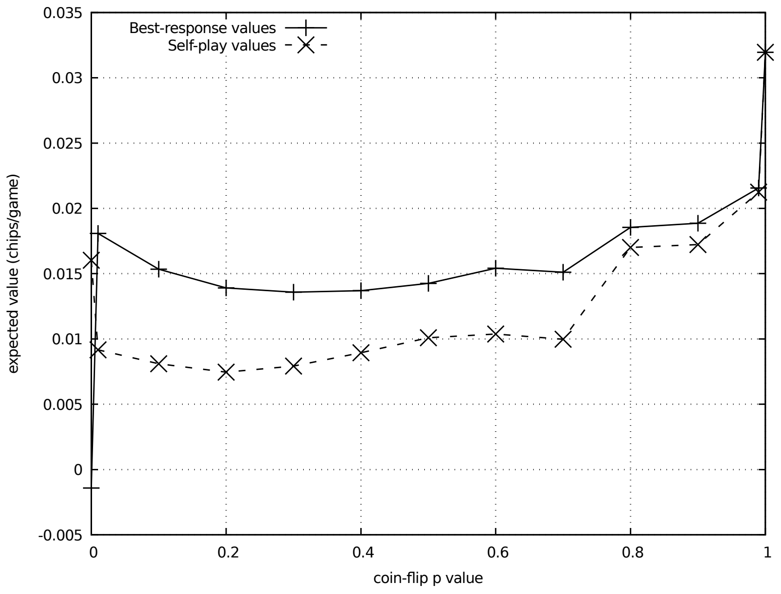


Figure 5.12: Performance of re-solved abstract strategy in Leduc hold'em against original strategy, using coin-flip gadget, with self-play and best-response values

Informally, we can explain the lack of increased exploitability by considering $1 - p$ as slack in the opponent probabilities. Without the coin-flip gadget, the opponent is able to pick any distribution of probabilities coming into the rest of the game. With the coin-flip gadget, this distribution is mixed with a fixed distribution. As long as $1 - p$ is small enough that the opponent can still select a distribution which is an appropriate response to our strategy when mixed with the fixed distribution, we have not unduly limited our opponent. If our opponent is not overly restricted, our strategy will not be exploitable.

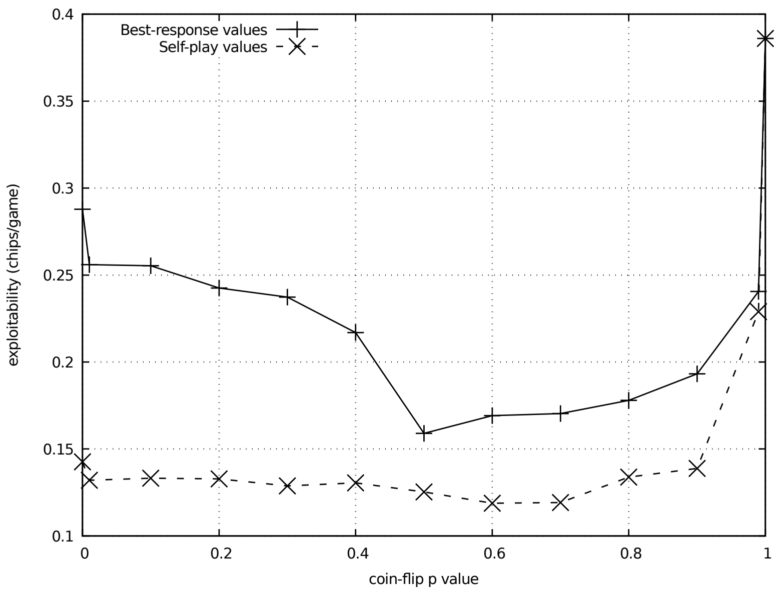


Figure 5.13: Exploitability of re-solved abstract strategy in Leduc hold'em, using coin-flip gadget, with self-play and best-response values

Alternate Re-solving Gadget

The gadget in Section 5.2.5 only guarantees that opponent counterfactual values do not increase after re-solving. If we are re-solving to reduce exploitability, we also have a goal of decreasing opponent counterfactual values. Moravčík *et al.* published a new re-solving gadget [66] which simultaneously guarantees opponent counterfactual values do not increase, while maximising the minimum improvement of the values.

Moravčík *et al.* make two changes in their gadget. First, the values at the gadget decision are shifted by subtracting the constraint values, so that a best response to the original strategy has a value of 0 for all opponent information sets. Second, the opponent chooses which information set to play (*i.e.,* one decision with N choices), rather than choosing whether to play each information set (*i.e.,* N binary decisions.)

The motivation behind the changes relies on the idea of margin: the difference between the opponent's counterfactual value for the re-solved strategy, and the opponent constraint values used to construct the gadget. By letting the opponent choose an information set, we get the minimum margins over all opponent information sets. Choosing our strategy to maximise our value means we are maximising the minimum margin, giving us incentive to decrease opponent values beyond the original constraint values.

While Moravčík *et al.* report positive results, the new gadget makes no difference within the experimental setup in Section 5.4.1. The lack of improvement does not contradict their reported results, but it does highlight a subtle distinction between maximising min-margin, and minimising the sum of opponent counterfactual values. If the constraint values are tight for one of the opponent information sets I, so that it cannot be decreased any further, the maximum margin will be 0. With an adequate strategy, all of the other margins will be at least 0. Because the opponent selects the information set with the minimum margin, they will select information set I and we will have no further incentive to improve.

Chapter 6

Practical Decomposition: Continual Re-Solving

The decomposition tools in Chapter 5 allowed for a new class of algorithms, which can operate on games which do not fit in memory. For example, there is enough space on modern machines to run a recursive version of CFR-D on the game of HUNL, which has more than $6 * 10^{161}$ information sets [44]. Unfortunately, the computation time is prohibitively large, and we would not complete a single iteration in anyone's lifetime[1]. In perfect information games, the computational intractability of extremely large games is solved by using depth-limited search, and a heuristic evaluation of positions at the depth limit.

CFR-D and the decomposition tools provide a framework for a depth-limited solving computation, using a game solving algorithm to answer questions about situations at the depth limit. This split gives us a location to switch out a slow computation for a fast heuristic evaluation function, just as in perfect information search, and defines the input and output of the evaluation function, but gives no guidance on how to construct an effective function. Martin Schmid and Matej Moravčík suggested the use of deep neural networks to build an evaluation function from training data for this purpose.

I participated in a large team effort to implement game-theoretically sound depth-limited solving in HUNL. We played the resulting program DeepStack against professional poker players, and won by a large, statistically signif-

[1]To continue the physics analogy of HUNL having more particles than the observable universe, the game-solving computation would take longer to finish than experimentally checking if the universe enters a big crunch or suffers from heat death.

icant margin. My main contribution is a theoretical analysis of the final algorithm done in cooperation with Trevor Davis. I also worked on coordination and infrastructure, and participated in general discussion throughout the project. The theoretical work, algorithm, and human results are published in Science [65].

6.1 Continual Re-solving Algorithm

DeepStack uses a new theoretically sound algorithm, continual re-solving, that performs multiple depth-limited computations to solve and play imperfect information games [65]. It combines elements of CFR-D and re-solving with a heuristic evaluation function. It is also an entirely online algorithm, with no pre-computed strategy.

As a basic outline, continuous re-solving uses CFR-D with an evaluation function to provide answers to the subgame solving problems of Section 5.2.2, and re-solves our strategy before every decision that we make using the technique of Section 5.2.5. There are two parts where continuous re-solving extends beyond these previously published techniques: the evaluation function, and the generation of opponent counterfactual values for re-solving constraints.

6.1.1 Imperfect Information Evaluation

At each iteration, CFR-D must solve every subgame given the fixed trunk strategy of the current iteration. This is an instance of the problem discussed in Section 5.2.2. We have some trunk strategy profile σ^T, which lets us compute the probability $\pi_p^{\sigma^T}(I_p)$ of reaching a subgame for both players p, and all information sets/parts[2] I_p at the root of the subgame. These probabilities make the subgame a well defined game which can be solved. Finally, the subgame equilibrium profile can be used to compute counterfactual values $v_p(I_p)$ for both players and all information sets/parts. Summarising the process for a subgame, we get a pair of probability distributions in, and produce a pair

[2]The partition of states at the root of the subgame for the non-acting player does not constitute a traditional information set partition, which only considers the acting player.

of counterfactual values corresponding to an equilibrium profile.

The summary gives us the form for the input and output of an imperfect information heuristic evaluation function, and the target values we are trying to estimate. Note that the form is similar to a perfect information heuristic function: we want the value of optimal play from the position onward, and as discussed before we only need $N-1$ elements of the distribution so that perfect information games require no distribution information. In contrast, there are a couple of differences that make it more difficult to construct an imperfect information heuristic. First, it has multiple real-valued inputs along with the situation itself, and has multiple correlated, real-valued outputs. Second, the counterfactual values of an equilibrium are not unique, so that it can be harder to evaluate performance.

The non-uniqueness of outputs for any given inputs means we do not strictly have a function, but any single choice of the possible outputs is a valid choice, so we can force it to be a function. The real-valued inputs rule out a table-based approach like the databases of endgame positions often used in perfect information search, but does not rule out any general techniques for function approximation. So we can produce an imperfect information heuristic evaluation function, using the well-developed body of research on function approximation.

It is interesting to note that in poker, there are transpositions in the evaluation function: different sequences of events that lead to equivalent situations with the same subgame structure and player utilities. All we need to know is how much each player has bet in total, how big of a bet they are currently facing, and whether the current round has already started with a check. While the exact betting sequence matters very much within the game and affects the future play, the effect of different betting sequences is to change the probability distribution of hands, and those distributions are already an input to the evaluation function. Instead of having $10^{161}/1326$ different subgames[3] we could consider evaluating in HUNL with 20 000 chip stacks with a 100 chip big

[3]There are around 10^{160} information sets, and all subgames group together the information sets for the 1326 possible hands.

blind, there are around 10^{18} unique subgame situations to evaluate[4].

6.1.2 Re-solving Constraint Values

The constraints used by the re-solving method in my original decomposition work are framed in terms of the best-response values to an existing strategy for the subgame of interest. In Section 5.4.2 I widened this to consider arbitrary constraint values. I showed that if we start with an exploitable strategy, best-response values are necessarily non-optimal, and experimentally demonstrated that self-play values can produce better results than best-response values. We could also consider different constraints for re-solving a subgame when our current strategy is not highly exploitable.

There are, in fact, a range of constraint values which preserve the original exploitability of σ when re-solving a subgame S. For example, if the opponent has made at least one action before reaching S, we can use the best-response values of not playing into S [40, 14]. Instead of doing as well as σ^S, we are now ensuring that we do as well as possible given an opponent that can either choose to play against our new strategy, or not play into S and play against the rest of σ. For information sets where the opponent will reach S in some equilibrium, these "anything-but-this-subgame" values will be a lower, more optimistic choice than the best-response counterfactual values.

For a sketch of a correctness proof for re-solving with these non-subtree values, we can consider the resulting gadget-game equilibrium profile within the original game. There is at least one opponent information set I before subgame S. The gadget decision is now making exactly the same decision as an opponent best response would make at I within the original game: deciding to play into S to receive the new value from σ'^S, or receive the value from playing a best response to σ outside of S. We are therefore trying to minimise our exploitability by changing σ'^S, given the fixed σ outside of S.

DeepStack also uses constraint values based on an opponent action at some information set I before subgame S. Instead of using the maximum value

[4]$20\,000^2/2$ combinations for total amount bet and bet faced, times 2 for the possibility of an initial check, times $2.4 * 10^9$ unique card combinations.

over actions at I that do not reach S, it uses the maximum value over all actions at I including the action which reaches S. That is, it is the opponent's optimal value at I. This produces constraint values which are a higher for any information set where reaching S was a non-optimal decision, making DeepStack's constraints a more pessimistic choice than the basic opponent best-response values.

The upside of using the opponent-optimal constraint values is that we only need a single value for each opponent information set, rather than requiring a value for each action. When using continual re-solving in a game like HUNL, we might use a sparse tree to find our current strategy. After making our decision and the opponent acts again, we might find out that the opponent's decision does not exactly match any of the choices in the sparse tree. However, as long as the sparse tree does a good job of estimating the opponent's value in an equilibrium (*i.e.,* their optimal value), we can safely use that single value per information set regardless of what action the opponent made.

Correctness of the opponent-optimal values follows from noting that the constraints must be feasible because no value is less than the self-play values, and applying Lemma 27 in Appendix C, originally published in the supplementary material accompanying the DeepStack article [65]. Informally, re-solving with DeepStack's choice of constraints is not guaranteed to exploit opponent mistakes as much as the original strategy, but we will never do worse than our original strategy would have done against an optimal opponent.

6.2 Theoretical Analysis

Along with Trevor Davis, I gave a bound on the exploitability of continual re-solving, stated below in Theorem 12. Given a game where we make d actions, the strategy has exploitability in $\mathcal{O}(d\epsilon_E)$ if we use a subgame evaluation function with bounded error ϵ_E. A proof is given in Appendix C.

Theorem 12 *Assume we have some initial opponent constraint values w from a solution generated using at least T iterations of CFR-D, we use at least T iterations of CFR-D to solve each re-solving game, and we use a subtree* *value estimator such that* $\min_{\sigma_S^* \in NE_S} \sum_{I \in \mathcal{I}_2^S} |v^{\sigma_S^*}(I) - v_I| \leq \epsilon_E$. *Then after d re-solving steps the exploitability of the resulting strategy is no more than* $(d+1)k/\sqrt{T} + (2d+1)j\epsilon_E$ *for some constants* k, j *specific to both the game and how it is split into subgames.*

While both continual re-solving and CFR-D are built on very similar decomposition tools, the proof of the DeepStack bound in Theorem 12 differs from the CFR-D bounds I published [15] in two main ways. First, the original CFR-D proof is very strongly tied to best-response constraints, whereas the continual re-solving proof needs to be more general. Second, the CFR-D proof used bounds on individual information sets, leading to a geometrically growing exploitability bound in the number of re-solving steps.

To handle DeepStack's constraints, the exploitability bounds were split into underestimation error, overestimation error, and solution error. This split made it easier to talk about the effects of the solution quality and evaluation function on constraints, and the effects of constraints on exploitability.

The original CFR-D proof gave bounds in terms of individual information sets at the root of the subgame. Looking at exploitability, we need to consider the error across all information sets, which meant the original CFR-D exploitability bound had an $\mathcal{O}(N\epsilon_E)$ term, where N was the number of information sets at the root of the subgame. Because the various sources of error were not separated, this term also became part of the input to the next re-solving step, resulting in an $\mathcal{O}(N^d\epsilon_E)$ bound for d re-solving steps.

The DeepStack proof gives a bound on the sum of errors across all information sets, giving $\mathcal{O}(\epsilon_E)$ terms without the factor of N. By more carefully separating the sources of error, the proof also avoids any other geometric growth in the d re-solving steps.

At each re-solving step the algorithm introduces error into the constraints by estimating the counterfactual values using the evaluation function, with the sum of the underestimation error and overestimation error being bounded by ϵ_E. There is also additional solution error from finding an approximate solution to the re-solving game. If we use a variant of CFR, the solution error is bounded by $\mathcal{O}(1/\sqrt{T} + \epsilon_E)$ when using T re-solving iterations. Running the re-solving step d times gives the final bound.

6.3 Experimental Results

There are two strong experimental results for DeepStack. First, we have lower bounds on exploitability which are significantly lower for DeepStack than for a variety of agents HUNL agents built using traditional abstraction-based techniques. Second, we have strong results playing DeepStack against humans. All of the results in this section were published as supplementary material in the DeepStack paper [65].

6.3.1 Lower Bounds on Exploitability

Lisý*et al.* introduced the local best response technique (LBR) [60] which provides a lower bound on exploitability of a static agent. LBR is not a single computation, but an agent that plays against the target agent using knowledge of the target's strategy to choose actions. At each decision, LBR uses the distribution over the opponent's possible hands to compute an approximation of future value for some set of actions. It then makes a greedy local decision by picking the action with the highest expected value.

Table 6.1 gives the LBR results against a number of agents including DeepStack, using four different configurations of LBR with different available actions. Slumbot and Act1 are agents from the 2016 Annual Computer Poker Competition (ACPC) [1], coming in second and third place respectively. Hyperborean was the third place instant run-off entry in the 2014 ACPC. The "Full Cards" agent uses a compressed 2TB strategy built from a game that uses the full HUNL card structure, but only allowed the actions of a fold, call, pot-sized bet, or an all-in bet. Unlike the ACPC agents and DeepStack, which use player stacks of 200 big blinds, the "Full Cards" strategy used player stacks of 100 big blinds.

LBR Settings	Local best response performance (mbb/g)			
Pre-flop	F, C	C	C	C
Flop	F, C	C	C	56bets
Turn	F, C	F, C, P, A	56bets	F, C
River	F, C	F, C, P, A	56bets	F, C
Hyperborean (2014)	721 ± 56	3852 ± 141	4675 ± 152	983 ± 95
Slumbot (2016)	522 ± 50	4020 ± 115	3763 ± 104	1227 ± 79
Act1 (2016)	407 ± 47	2597 ± 140	3302 ± 122	847 ± 78
Always Fold	250 ± 0	750 ± 0	750 ± 0	750 ± 0
Full Cards [100 BB]	-424 ± 37	-536 ± 87	2403 ± 87	1008 ± 68
DeepStack	-428 ± 87	-383 ± 219	-775 ± 255	-602 ± 214

Table 6.1: Exploitability lower bound of different programs evaluated with LBR using only the listed actions in each round as shown in each column. F, C, P, A, refer to fold, call, a pot-sized bet, and all-in, respectively. 56bets includes the actions fold, call and 56 equidistant pot fractions as defined in the original LBR paper [60].

All of the ACPC entries are extremely vulnerable to exploitation by LBR: they are worse off than an agent which folds to any bet. LBR does not even need to make a bet: just folding or calling with full knowledge of the cards is sufficient to win by hundreds of mbb/g. The "Full Cards" agent is similarly exploitable, as long as LBR is allowed to make actions other than the fold, call, pot, all-in actions that were used in the game for which "Full Cards" is a Nash equilibrium. In contrast, DeepStack cannot be exploited by LBR in any of the tested settings.

There are situations where allowing LBR to consider more actions does not increase its performance, because the greedy local decision might not be the best choice. Negative values in the LBR results are not a logical error either; they indicate that LBR was unable to exploit the agent because it is only estimating future values. Finally, note that LBR is only providing a lower bound on exploitability, so the negative values do not prove that DeepStack's exploitability is small. They do, however, indicate that DeepStack is not vulnerable to one class of attack that can be used to exploit a range of abstraction-based agents.

6.3.2 Human Performance against DeepStack

With the assistance of the International Federation of Poker(IFP) [39], we recruited volunteers to play against DeepStack. Only players who self-identified as a "professional poker player" during registration were selected to play. Each player was given four weeks to complete a 3000 game match. To incentivize players, monetary prizes of \$5000, \$2500, and \$1250 (CAD) were awarded to the top three players that completed their match. Player performance was ranked by their win rates, using an unbiased variance reduction technique called AIVAT [16]. All participants were informed of these details when they registered to participate, and required to acknowledge their understanding.

Table 6.2 gives the results of the human study. There were 33 players from 17 countries who participated, with 11 players completing all 3000 games. All games were completed between November 7th and December 12th, 2016. DeepStack won against all participants who completed their match, with a statistically significant win at more than a 95% confidence level for 10 of those 11 players. DeepStack did not lose at a statistically significant rate against any participant, and won an average of 486 mbb/g $\pm$ 40 against all players over the complete set of 44 852 games. The positive results make DeepStack the first computer agent to achieve a statistically significant win against professional poker players in the game of HUNL.

Player	Games	Win Rate (mbb/g)
Martin Sturc	3000	70 ± 119
Stanislav Voloshin	3000	126 ± 103
Prakshat Shrimankar	3000	139 ± 97
Ivan Shabalin	3000	170 ± 99
Lucas Schaumann	3000	207 ± 87
Phil Laak	3000	212 ± 143
Kaishi Sun	3000	363 ± 116
Dmitry Lesnoy	3000	411 ± 138
Antonio Parlavecchio	3000	618 ± 212
Muskan Sethi	3000	1009 ± 184
Pol Dmit	3000	1008 ± 156
Tsuneaki Takeda	1901	627 ± 231
Youwei Qin	1759	1306 ± 331
Fintan Gavin	1555	635 ± 278
Giedrius Talacka	1514	1063 ± 338
Juergen Bachmann	1088	527 ± 198
Sergey Indenok	852	881 ± 371
Sebastian Schwab	516	1086 ± 598
Dara O'Kearney	456	78 ± 250
Roman Shaposhnikov	330	131 ± 305
Shai Zurr	330	499 ± 360
Luca Moschitta	328	444 ± 580
Stas Tishekvich	295	−45 ± 433
Eyal Eshkar	191	18 ± 608
Jefri Islam	176	997 ± 700
Fan Sun	122	531 ± 774
Igor Naumenko	102	−137 ± 638
Silvio Pizzarello	90	1500 ± 2100
Gaia Freire	76	369 ± 136
Alexander Bös	74	487 ± 756
Victor Santos	58	475 ± 462
Mike Phan	32	−1019 ± 2352
Juan Manuel Pastor	7	2744 ± 3521

Table 6.2: DeepStack's win rate against professional poker players, with 95% confidence interval

Chapter 7

Conclusions

In this thesis I introduce improved methods for solving imperfect information games, trying to find techniques that use less computation time, less space, or both. Along with others, I introduced three new algorithms: CFR$^+$, CFR-D, and continual re-solving. I provide performance bounds for all three algorithms, and tighten an existing CFR bound as part of a larger effort to explain empirical performance compared to theoretically faster algorithms.

In experiments using large poker and matrix games, CFR$^+$ produces a low-exploitability strategy much more quickly than CFR. The behaviour of CFR$^+$ in these games is more like we would expect from theoretically faster algorithms. With others, I used CFR$^+$ to solve HULHE, making it the first solved imperfect information game that is commonly played by humans. I provide theoretical bounds which show that CFR$^+$ is asymptotically no worse than CFR. I also prove that regret-matching$^+$, a new algorithm used by CFR$^+$, minimises tracking regret, giving it a guarantee with respect to one class of dynamic policies. Hedge and regret-matching lack this property.

The theoretical decomposition tools I introduced opened up new space-saving possibilities for imperfect information games, and for improving existing strategies. CFR-D demonstrated some of this potential, with space requirements that are less than linear in the number of information sets. I prove that CFR-D converges, with a roughly quadratic computation cost compared to CFR. The CFR-D result rests on top of a proof that re-solving works, and can use opponent counterfactual values to compute a subgame strategy which completes a trunk equilibrium.

Continual re-solving extended the decomposition work, using a heuristic evaluation function to not only overcome the quadratic increase in computation time suffered by CFR-D, but greatly decrease the computation time compared to CFR. Along with the decomposition-based reduction in space requirements, and sparse game trees, the combination of advantages make it possible to approximately solve and play extremely large games. With others, I used continual re-solving to play and beat professional HUNL players for the first time. Trevor Davis and I proved that the exploitability of continual re-solving grows linearly with the heuristic error.

There are many possible directions to extend this research. I improved the bounds on CFR (and CFR$^+$) with respect to problem size, but they are still very loose. Decomposition is not directly tied to CFR, as in CFR-D or continual re-solving, but it is not clear how to fit the resulting non-smooth objective function into other algorithms. All of the fast $1/\epsilon$ algorithms seem to require 3 values per information set: is there a modification that only requires 2 values? Can we use the observation of Rakhlin *et al.* that the opponent is not arbitrary to improve on the $1/\sqrt{\epsilon}$ bounds for CFR$^+$, perhaps to $\epsilon^{-3/4}$? Finally, continual re-solving has many pieces that could be improved: better heuristics, different ways of building sparse depth-limited trees, and changing the re-solving method and how opponent ranges are used.

With or without future improvements, this work extended what was possible with imperfect information games. In games where space is not an issue, CFR$^+$ is simple and very effective, and can be scaled up to large clusters of machines. If there is not enough space but we do have time, CFR-D can solve games using less memory than conventional methods. Finally, if we have neither time nor space but can accept error introduced by heuristic evaluation, continual re-solving still lets us solve and play these games.

Bibliography

[1] Annual Computer Poker Competition. http://www.computerpokercompetition.org/index.php/about (Accessed March 13, 2017).

[2] Charles Babbage. *Passages from the Life of a Philosopher*. Longman, Green, Longman, Roberts, and Green, London, 1864. Chapter 34.

[3] Kaushik Basu. The traveler's dilemma: Paradoxes of rationality in game theory. *The American Economic Review*, 84(2):391–395, 1994.

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

[5] David Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956.

[6] Émile Borel and Jean Ville. *Applications de la théorie des probabilités aux jeux de hasard*. Gauthier-Villars, 1938.

[7] Branislav Bosanský, Albert Xin Jiang, Milind Tambe, and Christopher Kiekintveld. Combining compact representation and incremental generation in large games with sequential strategies. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 812–818, 2015.

[8] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. CFR$^+$. `http://poker.cs.ualberta.ca/software/CFR_plus.tar.bz2`.

[9] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.

[10] Michael H. Bowling, Nicholas Abou Risk, Nolan Bard, Darse Billings, Neil Burch, Joshua Davidson, John Alexander Hawkin, Robert Holte, Michael Johanson, Morgan Kan, Bryce Paradis, Jonathan Schaeffer, David Schnizlein, Duane Szafron, Kevin Waugh, and Martin Zinkevich. A demonstration of the polaris poker system. In *Proceedings of the Eigth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1391–1392, 2009.

[11] L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200 – 217, 1967.

[12] Jacob Bronowski. The ascent of man. Documentary, 1973. Episode 13.

[13] Noam Brown, Sam Ganzfried, and Tuomas Sandholm. Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit texas hold'em agent. In *Computer Poker and Imperfect Information, Papers from the 2015 AAAI Workshop*, 2015.

[14] Noam Brown and Tuomas Sandholm. A time and space efficient algorithm for approximately solving large imperfect information games. In *Proceedings of the Computer Poker and Imperfect Information Games Workshop at AAAI 2017*, 2017.

[15] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 602–608, 2014.

[16] Neil Burch, Martin Schmid, Matej Moravcik, and Michael Bowling. Aivat: A new variance reduction technique for agent evaluation in imperfect information games. In *AAAI Workshop on Computer Poker and Imperfect Information Games*, 2017. http://arxiv.org/abs/1612.06915.

[17] Nicolò Cesa-Bianchi, Yoav Freund, David Haussler, David P. Helmbold, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. *J. ACM*, 44(3):427–485, 1997.

[18] Kamalika Chaudhuri, Yoav Freund, and Daniel J Hsu. A parameter-free hedging algorithm. In *Advances in Neural Information Processing Systems 22 (NIPS)*, pages 297–305, 2009.

[19] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM (JACM)*, 56(3):14, 2009.

[20] George Dantzig. Constructive proof of the min-max theorem. *Pacific Journal of Mathematics*, 6(1):25–33, 1956.

[21] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[22] Olivier Devolder, François Glineur, and Yurii Nesterov. First-order methods of smooth convex optimization with inexact oracle. *Math. Program.*, 146(1-2):37–75, 2014.

[23] Robert Dorfman. Application of the simplex method to a game theory problem. In *Activity Analysis of Production and Allocation-Proceedings of a Conference*, pages 348–358, 1951.

[24] Fei Fang, Thanh Hong Nguyen, Rob Pickles, Wai Y. Lam, Gopalasamy R. Clements, Bo An, Amandeep Singh, and Milind Tambe. Deploying PAWS to combat poaching: Game-theoretic patrolling in areas with complex terrain (demonstration). In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 4355–4356, 2016.

[25] D.A. Ferrucci. Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1:1–1:15, May 2012.

[26] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[27] Sam Ganzfried and Tuomas Sandholm. Action translation in extensive-form games with large action spaces: Axioms, paradoxes, and the pseudo-harmonic mapping. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 120–128, 2013.

[28] Sam Ganzfried and Tuomas Sandholm. Improving performance in imperfect-information games with large state and action spaces by solving endgames. In *Procedings of the Computer Poker and Imperfect Information Workshop at AAAI 2013*, 2013.

[29] Sam Ganzfried and Tuomas Sandholm. Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 682–690, 2014.

[30] Richard G. Gibson, Neil Burch, Marc Lanctot, and Duane Szafron. Efficient monte carlo counterfactual regret minimization in games with many player actions. In *Advances in Neural Information Processing Systems 25 (NIPS)*, 2012.

[31] Andrew Gilpin, Samid Hoda, Javier Peña, and Tuomas Sandholm. Gradient-based algorithms for finding nash equilibria in extensive form games. In *Proceedings of the Eighteenth International Conference on Game Theory*, 2007.

[32] Andrew Gilpin and Toumas Sandholm. A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, 2006.

[33] Andrew Gilpin and Tuomas Sandholm. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence.* AAAI Press, 2007.

[34] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sorensen. A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *Proceedings of the Seventh international joint conference on Autonomous agents and multi-agent systems*, 2008.

[35] Jean-Louis Goffin. On convergence rates of subgradient optimization methods. *Mathematical programming*, 13(1):329–347, 1977.

[36] Timothy Hart and Daniel Edwards. The alpha-beta heuristic. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1963.

[37] Mark Herbster and Manfred K. Warmuth. Tracking the best expert. pages 286–294, 1995.

[38] Samid Hoda, Andrew Gilpin, Javier Pea, and Tuomas Sandholm. Smoothing techniques for computing nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010.

[39] International Federation of Poker. http://pokerfed.org/about/ (Accessed March 13, 2017).

[40] Eric Jackson. A time and space efficient algorithm for approximately solving large imperfect information games. In *Procedings of the Computer Poker and Imperfect Information Workshop at AAAI 2014*, 2014.

[41] Manish Jain, Dmytro Korzhyk, Ondrej Vanek, Vincent Conitzer, Michal Pechoucek, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 327–334, 2011.

[42] Albert Xin Jiang, Zhengyu Yin, Chao Zhang, Milind Tambe, and Sarit Kraus. Game-theoretic randomization for security patrolling with dynamic execution uncertainty. In *Proceedings of the Twelfth International conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 207–214, 2013.

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

[44] Michael Johanson. Measuring the size of large no-limit poker games. Technical report, University of Alberta, 2013.

[45] Michael Johanson, Nolan Bard, Neil Burch, and Michael Bowling. Finding optimal abstract strategies in extensive-form games. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[46] Michael Johanson, Neil Burch, Richard Valenzano, and Michael Bowling. Evaluating state-space abstractions in extensive-form games. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

[47] Michael Johanson, Kevin Waugh, Michael H. Bowling, and Martin 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.

[48] Michael Johanson, Martin Zinkevich, and Michael H. Bowling. Computing robust counter-strategies. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 721–728, 2007.

[49] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 302–311, New York, NY, USA, 1984. ACM.

[50] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Bahavior*, 4(4):528–552, 1992.

[51] Daphne Koller, Nimrod Megiddo, and Bernhard Von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14:247–259, 1996.

[52] David M Kreps and Robert Wilson. Sequential equilibria. *Econometrica: Journal of the Econometric Society*, pages 863–894, 1982.

[53] Christian Kroer and Tuomas Sandholm. Extensive-form game abstraction with bounds. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation (EC)*, pages 621–638, 2014.

[54] Christian Kroer, Kevin Waugh, Fatma Kilinç-Karzan, and Tuomas Sandholm. Faster first-order methods for extensive-form game solving. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation (EC)*, pages 817–834, 2015.

[55] Christian Kroer, Kevin Waugh, Fatma Kilinç-Karzan, and Tuomas Sandholm. Theoretical and practical advances on smoothing for extensive-form games. *CoRR*, abs/1702.04849, 2017.

[56] H.W. Kuhn. Simplified two-person poker. In H.W. Kuhn and A.W. Tucker, editors, *Contributions to the Theory of Games*, volume 1 of *Annals of mathematics studies*, pages 97–103. Princeton University Press, 1950.

[57] Marc Lanctot, Richard Gibson, Neil Burch, Marty Zinkevich, and Michael Bowling. No-regret learning in extensive-form games with imperfect recall. In *Proceedings of the Twenty-Ninth International Conference on Machine Learning*, 2012.

[58] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. Technical report, University of Alberta, 2009.

[59] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems 22 (NIPS)*, pages 1078–1086, 2009.

[60] Viliam Lisý and Michael Bowling. Eqilibrium approximation quality of current no-limit poker bots. In *AAAI Workshop on Computer Poker and Imperfect Information Games*, 2017. https://arxiv.org/abs/1612.07547.

[61] Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

[62] Haipeng Luo and Robert E Schapire. Achieving all with no parameters: Adaptive normalhedge. *arXiv preprint arXiv:1502.05934*, 2015.

[63] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *ICML*, pages 536–543, 2003.

[64] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[65] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[66] Matej Moravčík, Martin Schmid, Karel Ha, Milan Hladík, and Stephen J. Gaukrodger. Refining subgames in large imperfect information games. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 572–578, 2016.

[67] John F Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[68] Arkadi Nemirovski. Prox-method with rate of convergence o(1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

[69] A.S. Nemirovskiĭ and D.B. IUdin. *Problem Complexity and Method Efficiency in Optimization.* A Wiley-Interscience publication. Wiley, 1983.

[70] Y. Nesterov. Excessive gap technique for nonsmooth convex optimization. *SIAM Journal of Optimization*, 16(1):235–249, 2005.

[71] Yurii Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, 2005.

[72] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

[73] Alexander Rakhlin and Karthik Sridharan. Optimization, learning, and games with predictable sequences. In *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 3066–3074, 2013.

[74] RT Rockafellar. *Convex Analysis.* Princeton University Press, 1970.

[75] I. V. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3:678–681, 1962.

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

[77] Reinhard Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. *International journal of game theory*, 4(1):25–55, 1975.

[78] Claude E. Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[79] Jiefu Shi and Michael Littman. Abstraction methods for game theoretic poker. In *Computers and Games*, pages 333–345. Springer, 2001.

[80] NZ Shor, Krzysztof C Kiwiel, and Andrzej Ruszcayski. *Minimization methods for non-differentiable functions*. Springer-Verlag New York, Inc., 1985.

[81] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[82] Finnegan Southey, Michael H. Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and D. Chris Rayner. Bayes' bluff: Opponent modelling in poker. In *Proceedings of the Twenty-First Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 550–558, 2005.

[83] Vasilis Syrgkanis, Alekh Agarwal, Haipeng Luo, and Robert E. Schapire. Fast convergence of regularized learning in games. In *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 2989–2997, 2015.

[84] Oskari Tammelin. CFR+. *CoRR*, abs/1407.5042, 2014.

[85] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[86] John Tromp and Gunnar Farnebäck. Combinatorics of go. In *International Conference on Computers and Games*, pages 84–99. Springer, 2006.

[87] Paul Tseng. On accelerated proximal gradient methods for convex-concave optimization. `http://www.mit.edu/~dimitrib/PTseng/papers/apgm.pdf`, 2008.

[88] A. W. Tucker. Solving a matrix game by linear programming. *IBM J. Res. Dev.*, 4(5):507–517, November 1960.

[89] Eric Van Damme. A relation between perfect equilibria in extensive form games and proper equilibria in normal form games. *International Journal of Game Theory*, 13(1):1–13, 1984.

[90] J. von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[91] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, Second Edition, 1947.

[92] Kevin Waugh, Dave Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathologies in extensive games. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 781–788, 2009.

[93] Uzi Zahavi, Ariel Felner, Neil Burch, and Robert C. Holte. Predicting the performance of IDA* using conditional distributions. *Journal of Artificial Intelligence Research (JAIR)*, 37:41–83, 2010.

[94] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 905–912, 2007.

Appendix A

CFR$^+$ and Regret-matching$^+$ Proofs

In this section, I present proofs of the bounds given in Section 4.2. The proofs were originally published at IJCAI [85].

A.1 Regret-matching$^+$ and External Regret

The proof that regret-matching$^+$ has low external regret has two parts main. Lemma 13 shows that the regret-like values stored by the algorithm are an upper bound on the true regrets. Lemma 14 gives a bound on the regret-like values. Both lemmas together give the main result, Theorem 6.

Lemma 13 *Given a sequence of strategies $\sigma^1, ..., \sigma^T$, each defining a probability distribution over a set of actions A, let $Q^t(a) = \left(Q^{t-1}(a) + \Delta R^t(a)\right)^+$ and $Q^0(a) = 0$ for all actions $a \in A$. Then for the regret-like value $Q^t(a)$, we have*

$$\Delta R^t(a) = R^t(a) - R^{t-1}(a) \leq \Delta Q^t(a) = Q^t(a) - Q^{t-1}(a) \qquad \text{(A.1)}$$

and

$$R^t(a) \leq Q^t(a) \qquad \text{(A.2)}$$

Proof. For any $t \geq 1$ we have

$$\begin{aligned}\Delta Q^t(a) &= Q^t(a) - Q^{t-1}(a) \\ &= \max\left(Q^{t-1}(a) + \Delta R^t(a), 0\right) - Q^{t-1}(a)\end{aligned}$$

$$
\begin{aligned}
&\geq Q^{t-1}(a) + \Delta R^t(a) - Q^{t-1}(a) \\
&= \Delta R^t(a)
\end{aligned}
$$

After noting that $R^0(a) = 0 \leq Q^0(a) = 0$, we have proven equation A.2. Writing the regrets as sum of the changes, we can finish the proof of equation A.1.

$$
\begin{aligned}
Q^t(a) &= \sum_{i=1}^{t} \Delta Q^t(a) \\
&\geq \sum_{i=1}^{t} \Delta R^t(a) \\
&= R^t(a)
\end{aligned}
$$

■

Lemma 14 *Given a set of actions A and any sequence of T value functions $v^t : A \mapsto \mathbb{R}$, say we have played the sequence σ^t of regret-matching$^+$ strategies. Then the sum of squared regret-like values is bounded by the sum of squared change in regret values, so that $\sum_a Q^T(a)^2 \leq \sum_{t=1}^{T} \sum_a \Delta R^t(a)^2$.*

Proof.

$$
\begin{aligned}
\sum_a Q^T(a)^2 &= \sum_a \left(\left(Q^{T-1}(a) + \Delta R^T(a) \right)^+ \right)^2 \\
&\leq \sum_a \left(Q^{T-1}(a) + \Delta R^T(a) \right)^2 \\
&= \sum_a \left(Q^{T-1}(a)^2 + \Delta R^T(a)^2 + 2Q^{T-1}(a) \Delta R^T(a) \right) \\
&= \sum_a Q^{T-1}(a)^2 + \sum_a \Delta R^T(a)^2 \\
&\quad + 2 \sum_a Q^{T-1}(a) \left(v^T(a) - \sum_b v^T(b) \sigma^T(b) \right) \\
&= \sum_a Q^{T-1}(a)^2 + \sum_a \Delta R^T(a)^2 \\
&\quad + 2 \sum_a Q^{T-1}(a) \left(v^T(a) - \sum_b v^T(b) Q^{T-1}(b) / \sum_c Q^{T-1}(c) \right) \\
&= \sum_a Q^{T-1}(a)^2 + \sum_a \Delta R^T(a)^2 \\
&\quad + 2 \sum_a Q^{T-1}(a) v^T(a)
\end{aligned}
$$

$$\begin{aligned}
&- 2\sum_{a,b} Q^{T-1}(a)v^T(b)Q^{T-1}(b)/\sum_c Q^{T-1}(c) \\
&= \sum_a Q^{T-1}(a)^2 + \sum_a \Delta R^T(a)^2 \\
&\quad + 2\sum_a Q^{T-1}(a)v^T(a) \\
&\quad - 2\sum_b v^T(b)Q^{T-1}(b)\sum_a \sigma^T(a) \\
&= \sum_a Q^{T-1}(a)^2 + \sum_a \Delta R^T(a)^2 \\
&\quad + 2\sum_a Q^{T-1}(a)v^T(a) \\
&\quad - 2\sum_b v^T(b)Q^{T-1}(b)1 \\
&= \sum_a Q^{T-1}(a)^2 + \sum_a \Delta R^T(a)^2
\end{aligned}$$

$Q^0(a) = 0$ for all a, so by induction $\sum_a Q^T(a)^2 \leq \sum_{t=1}^T \sum_a \Delta R^t(a)^2$. ■

Theorem 6 *Given a set of actions A, any sequence of T value functions $v^t : A \mapsto \mathbb{R}$, and bound L such that $|v^t(a) - v^t(b)| \leq L$ for all t and $a, b \in A$, after playing the sequence σ^t of regret-matching$^+$ strategies, the regret-like value $Q^T(a) \leq L\sqrt{|A|T}$ for all $a \in A$. Moreover, this same bound applies to $R^T(a)$.*

Proof. For all t, $\Delta R^t(a) = v^t(a) - \sum_b v^t(b)\sigma^t(b) \leq L$. Placing this in Lemma 14 gives a bound $\sum_a Q^T(a)^2 \leq T|A|L^2$. Because $Q^T(a) \geq 0$ for all T, a for any a we get that $Q^T(a) \leq L\sqrt{|A|T}$. ■

A.2 Regret-matching$^+$ and Tracking Regret

To prove a tracking regret bound on regret-matching$^+$, we show that the regret-like values in regret-matching$^+$ are an upper bound on the true regret values within any of the piecewise segments.

Theorem 7 *Given a set of actions A, any sequence of T value functions $v^t : A \mapsto \mathbb{R}$, and bound L such that $|v^t(a) - v^t(b)| \leq L$ for all t and $a, b \in A$, after playing the sequence σ^t of regret-matching$^+$ strategies, there is a bound on k-tracking regret of $R^{k,T} \leq kL\sqrt{|A|T}$.*

Proof. Given a run of regret-matching$^+$ on the T time steps, we get a sequence of $Q^t(a)$ and $\Delta Q^t(a)$ values. Define $Q^{[B]}(a) = \sum_{t=B_\mathrm{s}}^{B_\mathrm{e}} \Delta Q^t(a)$, similar to the definition of $R^{[B]}(a)$ in equation 4.1 of Section 4.2.2. Then,

$$
\begin{aligned}
\sum_a \left(R^{[B]}(a)^+\right)^2 &= \left(\left(\sum_{t=B_\mathrm{s}}^{B_\mathrm{e}} \Delta R^t(a)\right)^+\right)^2 \\
&\leq \sum_a \left(\left(\sum_{t=B_\mathrm{s}}^{B_\mathrm{e}} \Delta Q^t(a)\right)^+\right)^2 && \text{Lemma 13} \\
&\leq \sum_a \left(\left(Q^{B_s-1}(a) + \sum_{t=B_\mathrm{s}}^{B_\mathrm{e}} \Delta Q^t(a)\right)^+\right)^2 && Q^t(a) \geq 0 \forall t \\
&= \sum_a \left(Q^{B_e}(a)^+\right)^2 \\
&= \sum_a Q^{B_e}(a)^2 \\
&\leq \sum_{t=1}^{B_e} \sum_a \Delta R^t(a)^2 && \text{Lemma 14} \\
&\leq B_e |A| L^2
\end{aligned}
\tag{A.3}
$$

Consider an arbitrary partition $\mathcal{B}$ of $[1, T]$ where $|\mathcal{B}| \leq k$. Compute the square of positive regrets over all intervals and piecewise strategies consistent with $\mathcal{B}$,

$$
\begin{aligned}
\left(\sum_{B \in \mathcal{B}} \max_{a_B} R^{[B]}(a_B)^+\right)^2 &\leq \left(|\mathcal{B}| \max_{B \in \mathcal{B}} \max_a R^{[B]}(a)^+\right)^2 \\
&\leq |\mathcal{B}|^2 \max_{B \in \mathcal{B}} \max_a \left(R^{[B]}(a)^+\right)^2 \\
&\leq k^2 \max_{B \in \mathcal{B}} \sum_a \left((R^{[B]}(a))^+\right)^2 \\
&\leq k^2 \max_{B \in \mathcal{B}} B_\mathrm{e} |A| L^2 && \text{Equation A.3} \\
&= k^2 L^2 |A| T
\end{aligned}
\tag{A.4}
$$

Comparing regrets and positive regrets,

$$
\begin{aligned}
\sum_{B \in \mathcal{B}} \max_{a_B} R^{[B]}(a_B) &\leq \sum_{B \in \mathcal{B}} \max_{a_B} R^{[B]}(a_B)^+ \\
&\leq kL\sqrt{|A|T} && \text{Equation A.4}
\end{aligned}
$$

Because $\mathcal{B}$ is an arbitrary k-partition, $R^{k,T} \leq kL\sqrt{|A|T}$. ■

A.3 CFR$^+$ Exploitability

With Theorem 6 giving a proof that regret-matching$^+$ minimises regret, proving that CFR$^+$ generates an ϵ-Nash equilibrium is largely achieved by existing work on CFR. The two major steps are linking the CFR$^+$ weighted average strategy and the regret over a weighted series of values, and providing a bound on regret-matching$^+$ regret on the weighted series of values.

We start with Lemma 15, which considers sequences which could plausibly be a sequence of $\Delta Q^t(a)$ values observed during a run of regret-matching$^+$. The lemma states that for any sequence of T bounded values where the sum of any prefix is positive, the linearly weighted sum of the values is less than T times greater than the un-weighted sum. A rough sketch of the argument is that the weighted sum is maximised by having positive values later in the sequence, but the positive-prefix-sum requirement places a bound on the suffix.

We then construct a new regret minimising problem from a CFR$^+$ run by taking the observed counterfactual values and using the same linearly increasing weighting CFR$^+$ uses for the average strategy. In Lemma 16 we use Lemma 15 and the proof of Theorem 1 to get a bound on the regret for the CFR$^+$ strategies in the new weighted problem. The (re-scaled) regret of a sequence of strategies in this new game gives the difference in value between a best response to the CFR$^+$ average strategy and the weighted observed values. Theorem 8 uses this link between regret and the CFR$^+$ average strategy.

Lemma 15 *Call a sequence $x_1, ..., x_T$ of bounded real values B-plausible if $B > 0$, $\sum_{t=1}^{i} x_t \geq 0$ for all i, and $\sum_{t=1}^{T} x_t \leq B$. For any B-plausible sequence, $\sum_{t=1}^{T} tx_t \leq TB$.*

Proof. Consider any B-plausible sequence that maximises the weighted sum. That is, let

$$x_1^*, ..., x_T^* = \operatorname*{argmax}_{B\text{-plausible } x_1', ..., x'^T} \sum_{t=1}^{T} tx_t'$$

We will show $\sum_{t=1}^{T} tx_t^* \leq TB$ by proving that $x_i^* \geq 0$ for all $1 \leq i \leq T$.

For any $i < T$, it cannot be the case that for any $i < j$ that $x_i^* > 0$, $x_j^* < 0$, and $x_k^* \geq 0$ for all k where $i < k < j$. Assume this were true, then let $\delta = \min(|x_i^*|, |x_j^*|)$. Construct a new sequence x' where

$$\begin{aligned} x_i' &= x_i^* - \delta \\ x_j' &= x_j^* + \delta \\ x_k' &= x_k^* && \forall k \neq i, j \end{aligned}$$

This new sequence x' is B-plausible. We have

$$\begin{aligned} \sum_{t=1}^{k} x_t' &= -\delta + \sum_{t=1}^{i-1} x_t^* + \sum_{t=i}^{k} x_t^* && \text{If } i \leq k < j \\ &\geq -\delta + 0 + \delta \\ &= 0 \end{aligned}$$

and

$$\sum_{t=1}^{k} x_t' = \sum_{t=1}^{k} x_t^* \qquad \text{If } k < i \text{ or } k \geq j$$

Looking at the weighted sum of x', we have

$$\begin{aligned} \sum_{t=1}^{T} t x_t' &= \sum_{t=1}^{i-1} t x_t' + + i x_i' + \sum_{t=i+1}^{j-1} t x_t' + j x_j' + \sum_{t=j+1}^{T} t x_t' \\ &= \sum_{t=1}^{i-1} t x_t^* + i(x_i^* - \delta) + \sum_{t=i+1}^{j-1} t x_t^* + j(x_j^* + \delta) + \sum_{t=j+1}^{T} t x_t^* \\ &= \sum_{t=1}^{T} t x_t^* + (j - i)\delta \\ &> \sum_{t=1}^{T} t x_t^* \end{aligned}$$

which contradicts the construction of x^* as a maximizing sequence.

Further, it cannot be the case that $x_j^* < 0$ for any j. Assume there is a negative value. Let j be the minimum index such that that $x_j^* < 0$. Because j is the minimum index, $x_k^* \geq 0$ for all $k < j$. From above, it cannot be the case that $x_i^* > 0$ for any $i < j$. This means $x_k^* = 0$ for all $k < j$, so we have $\sum_{t=1}^{j} x_t^* = x_j^* < 0$, which contradicts x^* being a B-plausible sequence. Therefore, we have $x_j^* \geq 0$ for all j.

Finally, looking at $\sum_{t=1}^{T} tx_t$ for an arbitrary B-plausible sequence x

$$\begin{aligned}
\sum_t tx_t &\leq \sum_t tx_t^* && \text{By construction of } x_t^* \\
&\leq \sum_t Tx_t^* && x_t^* \geq 0 \\
&= T\sum_t x_t^* \\
&\leq TB && x_t^* \text{ is } B\text{-plausible, so } \sum_t x_t^* \leq B
\end{aligned}$$

■

Lemma 16 *Let $\sigma^1, ..., \sigma^T$ be a sequence of strategy profiles from a run of CFR$^+$. Further, let L be a bound on the difference in values so that $|u_p(h) - u_p(j)| \leq L$ for all h, j, p, $A_p = \max_{I \in \mathcal{I}_p} |A(I)|$ be the maximum number of player p actions, and M_p be defined as in Theorem 1. Then for any player p, $R_p'^T = \max_{\sigma_p^*} \sum_{t=1}^{T} (tu_p(\sigma_p^*, \sigma_{-p}^t) - tu_p(\sigma_p^t, \sigma_{-p}^t)) \leq TLM_p\sqrt{A_pT}$.*

Proof. Consider the regret $R'^T(I,a)$ for σ^t in a new game where $u_p^t(h) = tu_p(h)$ for all h. By definition, $R'^T(I,a) = \sum_{t=1}^{T} t\Delta R^t(I,a)$. Lemma 13 and Lemma 15 let us say

$$R'^T(I,a) \leq \sum_{t=1}^{T} t\Delta Q^t(I,a) \leq T\sum_{t=1}^{T} \Delta Q^t(I,a) = TQ^T(I,a)$$

Lemma 14 lets us state that $\sum_a Q^T(I,a)^2 \leq \sum_{t=1}^{T} \sum_a \Delta R^t(I,a)^2$. This can be combined with the bound on $R'^T(I,a)$

$$\sum_a R'^T(I,a)^2 \leq \sum_a (TQ^T(I,a))^2 \leq \sum_{t=1}^{T} \sum_a \Delta(TR^t(I,a))^2$$

The bound on the sum on the squared regrets in the weighted game is in the same form as the regret-matching bound, and we can directly apply the proof of Theorem 1 in the modified game with $L' = TL$, giving us the bound $R_p \leq L'M_p\sqrt{A_pT} = TLM_p\sqrt{A_pT}$. ■

Theorem 8 *Let L be a bound on values such that $|u_p(h) - u_p(j)| \leq L$ for all h, j, p, and $A_p = \max_{I \in \mathcal{I}_p} |A(I)|$ be the maximum number of player p actions. Say we are given partitions $\{\mathcal{B}_p^0(\sigma_p^D), \mathcal{B}_p^1(\sigma_p^D), ...\}$ for any pure strategy $\sigma_p^D \in \Sigma_p^D$. Then after T CFR$^+$ iterations, the linearly weighted average strategy profile $\bar{\sigma}_p^T = 2/(T^2+T)\sum_{t=1}^T t\sigma_p^t$ is a $2\sum_p LM_p\sqrt{A_p/T}$-Nash equilibrium, where $M_p = \max_{\sigma_p^D \in \Sigma_p^D} \sum_{d=0}^{d_{\max}} \sum_{B \in \mathcal{B}_p^d(\sigma_p^D)} \xi_p(B)\sqrt{|B|}$.*

Proof. Let $z = \sum_{t=1}^T t = t(t+1)/2$. We wish to find the exploitability ϵ.

$$\epsilon = \max_{\sigma_1^*} u_1(\sigma_1^*, \frac{1}{z}\sum_t t\sigma_2^t) + \max_{\sigma_2^*} u_2(\frac{1}{z}\sum_t t\sigma_1^t, \sigma_2^*)$$

Because the game is zero sum, $\frac{1}{z}\sum_{t=1}^T tu_1(\sigma_1^t, \sigma_2^t) + \frac{1}{z}\sum_{t=1}^T tu_2(\sigma_1^t, \sigma_2^t) = 0$

$$= \max_{\sigma_1^*} u_1(\sigma_1^*, \frac{1}{z}\sum_t t\sigma_2^t) - \frac{1}{z}\sum_{t=1}^T tu_1(\sigma_1^t, \sigma_2^t)$$
$$+ \max_{\sigma_2^*} u_2(\frac{1}{z}\sum_t t\sigma_1^t, \sigma_2^*,) - \frac{1}{z}\sum_{t=1}^T tu_2(\sigma_1^t, \sigma_2^t)$$

Using Lemma 16 we can write this as

$$\begin{aligned}
&= \frac{1}{z}R_1'^T + \frac{1}{z}R_2'^T \\
&\leq \frac{1}{z}\sum_p TLM_p\sqrt{A_pT} \\
&\leq 2\sum_p LM_p\sqrt{A_p/T}
\end{aligned}$$

■

Appendix B

Decomposition Proofs

In this section, I present proofs of the arguments given in Section 5.2.3 and Section 5.2.5. The proofs were originally published at AAAI [15]. Some additional background terminology from the paper follows below.

In a perfect recall game, for any $z \in Z(I)$ there is a unique state $h \in I$ such that $h \sqsubset z$, which we write $z[I]$. If we replace the whole strategy for player p by a new strategy σ'_p, we will call the resulting profile $\langle \sigma_{-p}, \sigma'_p \rangle$. Finally, $\sigma_{[S \leftarrow \sigma']}$ is the strategy that is equal to σ everywhere except at information sets in S, where it is equal to σ'.

$\mathrm{BR}_p(\sigma)$ is the best response strategy for player p against σ. A counterfactual best response $\mathrm{CBR}_p(\sigma)$ is a strategy where $\sigma_p(I, a) > 0$ if and only if $v_p(I, a) \geq \max_b v_p(I, b)$, so it maximizes counterfactual value at every information set. CBR_p is necessarily a best response, but BR_p may not be a counterfactual best response as it may choose non-maximizing actions where $\pi_p(I) = 0$. The well known recursive bottom-up technique of constructing a best response generates a counterfactual best response.

A $\tilde{}$ will be used to distinguish the re-solving game from the subgame of the original game, and a re-solving strategy from the strategy copied to the original game. The strategy σ^S will refer to a strategy within subgame S in the original game. The game $\tilde{S}$ will be the re-solving game for S including the initial gadget decision. Strategy $\tilde{\sigma}$ is a strategy within the re-solving game $\tilde{S}$.

B.1 CFR-D exploitability

Theorem 17 gives a bound on regret after running CFR-D, based on the regret for each information set in the trunk due to regret-matching, and on the regrets $R^t_{full}(I) = \max_{\sigma'} v_p^{\sigma^t_{[\mathcal{I}_S \leftarrow \sigma']}}(I) - v_p^{\sigma^t}(I)$ for information sets $I \in \mathcal{I}_S$ at the root of each subgame S, given the subgame solution at time t. If there are at most N_S information sets at the root of any subgame and $R^t_{full}(I) \leq \epsilon_S$, then for any $\epsilon > 0$ CFR-D is guaranteed to produce an $\epsilon + N_S\epsilon_S$-Nash equilibrium with sufficient iterations. That is, the quality of the subgame solutions has a linear effect on the exploitability of the final CFR-D solution.

Theorem 17 *Let $\mathcal{I}_{TR}$ be the information sets in the trunk, $A = \max_{I \in \mathcal{I}} |A(I)|$ be an upper bound on the number of actions, and $\Delta = \max_{s,t \in Z} |u(s) - u(t)|$ be the variance in leaf utility. Let σ^t be the current CFR-D strategy profile at time t, and N_S be the number of information sets at the root of any subgame. If for all times t, players p, and information sets I at the root of a subgame $\mathcal{I}_S$, the regret $R^t_{full}(I) = \max_{\sigma'} v_p^{\sigma^t_{[\mathcal{I}_S \leftarrow \sigma']}}(I) - v_p^{\sigma^t}(I)$ is bounded by ϵ_S, then player p regret $R^T_p \leq \Delta|\mathcal{I}_{TR}|\sqrt{AT} + N_S T \epsilon_S$.*

Proof. The proof follows from Zinkevich *et al.*'s argument in Appendix A.1 [94]. Lemma 5 shows that for any player p information set I,

$$R^T_{full}(I) \leq R^T_p(I) + \sum_{I' \in Child_p(I)} R^T_{full}(I')$$

where $Child_p(I)$ is the set of all player information sets which can be reached from I without passing through another player p information set. We will prove $R^T_{full}(I) \leq (D(I)+1)\Delta\sqrt{AT} + S(I)T\epsilon_S$ by induction on $D(I)$, the number of I's descendants in the trunk $\mathcal{I}_{TR}$

$$D(I) = |\{J \in \mathcal{I}_{TR} | p(J) = P(I), \exists h \in I, j \in J \text{ s.t. } h \sqsubset j\}|$$

where $S(I)$ is the number of I's descendants in the subgame

$$S(I) = |\{J \in \mathcal{I}_S | p(J) = P(I), \exists h \in I, j \in J \text{ s.t. } h \sqsubset j\}|$$

As the base case, consider any trunk information set $I \in \mathcal{I}_p$ where $D(I) = 0$ (*i.e.,* no descendants in $\mathcal{I}_{TR}$). By Lemma 5 of Zinkevich *et al.*, we get

$$\begin{aligned}
R^T_{full}(I) &\leq R^T_p(I) + \sum_{I' \in Child_p(I)} R^T_{full}(I') \\
&\leq \Delta\sqrt{AT} + \sum_{I' \in Child_p(I)} R^T_{full}(I') && \text{By regret-matching bound} \\
&\leq \Delta\sqrt{AT} + S(I)\sum_{t=1}^{T} \epsilon_S && \text{By subgame regret bound} \\
&= (D(I)+1)\Delta\sqrt{AT} + S(I)T\epsilon_S
\end{aligned}$$

Now assume that for all $i \geq 0$, $R^T_{full}(I) \leq D(I)\Delta\sqrt{AT} + S(I)T\epsilon_S$ for all I such that $D(I) \leq i$. Consider $I \in \mathcal{I}_p$ such that $D(I) = i + 1$.

$$\begin{aligned}
R^T_{full}(I) &\leq R^T_p(I) + \sum_{I' \in Child_p(I)} R^T_{full}(I') \\
&\leq \Delta\sqrt{AT} + \sum_{I' \in Child_p(I)} D(I')(\Delta\sqrt{AT} + S(I')T\epsilon_S) \\
&= \Delta\sqrt{AT} + D(I)\sqrt{AT} + S(I)T\epsilon_S \\
&= (D(I)+1)\Delta\sqrt{AT} + S(I)T\epsilon_S
\end{aligned}$$

implying the inductive hypothesis holds for $i + 1$.

By induction $R^T_{full}(I) \leq (D(I)+1)\Delta\sqrt{AT} + S(I)T\epsilon_S$ holds for all I, and therefore holds at the root of the game where $D(I) = |\mathcal{I}_{TR}|$ and $S(I) = N_S$. ■

B.2 Re-solving Exploitability

In this section, I prove that in the worst case, re-solving a subgame only increases exploitability by a small, bounded amount, even if we started with an approximation of a Nash equilibrium. Without loss of generality, I will assume the players are 1 and 2, and we are re-solving to find a player 1 subgame strategy. We will use the re-solving gadget game of Section 5.2.5, so player 2 has the initial F and T decisions. The central theorem is

Theorem 18 *Let σ be a equilibrium profile approximation, with a bound on player 2 counterfactual regret $\max_{a\in(I)} v_2^g(I,a) - \sum_{b\in A(I)} \sigma_2(I,b)v_2^g(I,b) \leq \epsilon_S$ for all I in $\mathcal{I}_2^{\tilde{R}}$. Let $\tilde{\sigma}$ be the re-solved strategy, with a bound ϵ_R on the exploitability in the re-solving game. Then the exploitability of σ is increased by no more than $(|\mathcal{I}_2^{\tilde{R}}|-1)\epsilon_S + \epsilon_R$ if we use $\tilde{\sigma}$ in the subgame:*

$$u_2^{\langle \sigma_{1[S\leftarrow\tilde{\sigma}]}, BR(\sigma_{1[S\leftarrow\tilde{\sigma}]})\rangle} \leq (|\mathcal{I}_2^{\tilde{R}}|-1)\epsilon_S + \epsilon_R + u_2^{\langle \sigma_1, BR(\sigma_1)\rangle}$$

Theorem 18 gives a proof of the upper bound on exploitability of a re-solved strategy. The context for this section is as follows. Strategy profile σ is an approximation of a Nash equilibrium for the whole game. ϵ_S gives a bound on the regret-like value which is the difference between the maximum counterfactual value for any action at an information set, and the counterfactual value of the information set following σ. Strategy profile $\tilde{\sigma}$ is an ϵ_R-Nash equilibrium in the re-solving game. k is the normalising constant for the re-solving game, discussed in Section 5.2.5.

The proof will require some additional notation. The induced re-solving game strategy profile $\sigma^{\tilde{F}}$ is the re-solving game analogue of σ, with player 2 always choosing to play into the game, rather than terminating with the T action. More precisely, for all information sets in the subtrees under the F action, $\sigma^{\tilde{F}}$ takes the same action as σ, and for all initial player 2 information sets I where F or T is chosen, $\sigma^{\tilde{F}}(I,F)=1$.

Lemma 19 *For any player 2 strategy ρ in the original game and player 1 strategy $\tilde{\rho}$ in the re-solving game, if we let $\hat{\sigma} = \langle \sigma_{1[S\leftarrow\tilde{\rho}]}, \rho\rangle$, then for any $I \in \mathcal{I}_2^{\tilde{R}}$, $u_2^{\hat{\sigma}}(I) = \pi_2^{\rho}(I)\tilde{u}_2^{\langle\tilde{\rho},\rho^{\tilde{F}}\rangle}(I)$.*

Proof.

From the definition of expected value

$$u_2^{\hat{\sigma}}(I) = \sum_{h\in I}\sum_{z\in Z, h\sqsubset z} \pi_2^{\rho}(h)\pi_{-2}^{\sigma}(h)\pi_2^{\rho}(z|h)\pi_{-2}^{\tilde{\rho}}(z|h)u_2(z)$$

Noting $\pi_2^{\rho}(h) = \pi_2^{\rho}(I)$, and multiplying by k/k

$$= \pi_2^{\rho}(I)\sum_{h\in I}\sum_{z\in Z, h\sqsubset z} k\pi_{-2}^{\sigma}(h)\pi_2^{\rho}(z|h)\pi_{-2}^{\tilde{\rho}}(z|h)u_2(z)/k$$

Grouping $k\pi^{\sigma}_{-2}(h)$ into re-solving game probabilities, and $1/k$ into values

$$
\begin{aligned}
&= \pi^{\rho}_{2}(I) \sum_{h \in I} \sum_{z \in Z, h \sqsubset z} \pi^{\sigma^{\tilde{F}}}_{-2}(h) \pi^{\rho}_{2}(z|h) \pi^{\tilde{\rho}}_{-2}(z|h) \tilde{u}_2(z) \\
&= \pi^{\rho}_{2}(I) \tilde{u}_2^{\langle \tilde{\rho}, \rho^{\tilde{F}} \rangle}(I)
\end{aligned}
$$

■

Lemma 20 *Assume we are given an ϵ_R-Nash profile $\tilde{\sigma}$ in the re-solving game, a strategy profile σ in the original game with $v_2^{\langle \sigma_1, BR(\sigma_1) \rangle}(I) \leq \epsilon_S + v_2^{\sigma}(I)$, and a vector of weights w indexed by opponent information set with $0 \leq w_I \leq 1$ for all $I \in \mathcal{I}_2^{\tilde{R}}$. Then*

$$
\sum_{I \in \mathcal{I}_2^{\tilde{R}}} w_I \tilde{u}_2^{\langle \tilde{\sigma}_1, BR(\tilde{\sigma}_1) \rangle}(I) \leq (|\mathcal{I}_2^{\tilde{R}}| - 1)\epsilon_S + \epsilon_R + \sum_{I \in \mathcal{I}_2} w_I \tilde{u}_2^{\langle \sigma_1^{\tilde{F}}, BR(\sigma_1^{\tilde{F}}) \rangle}(I)
$$

Proof. σ and $\tilde{\sigma}$ have the following properties.

$$
\begin{aligned}
v_2^{\sigma}(I) &\leq v_2^{\langle \sigma_1, BR(\sigma_1) \rangle}(I) && \text{By def'n of BR} && \text{(B.1)} \\
&= \tilde{u}_2^{\langle \sigma_1^{\tilde{F}}, BR(\sigma_1^{\tilde{F}}) \rangle}(I) && \text{Gadget doesn't help 2} \\
&\leq \epsilon_S + v_2^{\sigma}(I) && \text{By } \epsilon_S \text{ bound on } \sigma && \text{(B.2)} \\
v_2^{\sigma}(I) &\leq \tilde{u}_2^{\langle \tilde{\sigma}_1, BR(\tilde{\sigma}_1) \rangle}(I) && \text{By gadget } T \text{ choice} && \text{(B.3)} \\
\sum_{I \in \mathcal{I}_2^{\tilde{R}}} \tilde{u}_2^{\langle \tilde{\sigma}_1, BR(\tilde{\sigma}_1) \rangle}(I) &\leq \epsilon_R + \sum_{I \in \mathcal{I}_2^{\tilde{R}}} \tilde{u}_2^{\tilde{\sigma}^*}(I) && \tilde{\sigma} \text{ an } \epsilon_R\text{-Nash} \\
&\leq \epsilon_R + \sum_{I \in \mathcal{I}_2^{\tilde{R}}} \tilde{u}_2^{\langle \sigma_1^{\tilde{F}}, BR(\sigma_1^{\tilde{F}}) \rangle}(I) && \sigma_1^{\tilde{F}} \text{ may not be Nash} && \text{(B.4)}
\end{aligned}
$$

That is, for each I we have a shared lower bound on $\tilde{u}_2^{\langle \sigma_1^{\tilde{F}}, BR(\sigma_1^{\tilde{F}}) \rangle}(I)$ and $\tilde{u}_2^{\langle \tilde{\sigma}_1, BR(\tilde{\sigma}_1) \rangle}(I)$, but only $\tilde{u}_2^{\langle \sigma_1^{\tilde{F}}, BR(\sigma_1^{\tilde{F}}) \rangle}(I)$ has an upper bound for each I. The upper bound for $\tilde{u}_2^{\langle \tilde{\sigma}_1, BR(\tilde{\sigma}_1) \rangle}(I)$ is only for the sum across all I.

To simplify, let $n(I) = \tilde{u}_2^{\langle \tilde{\sigma}_1, BR(\tilde{\sigma}_1) \rangle}(I)$, and $o(I) = \tilde{u}_2^{\langle \sigma_1^{\tilde{F}}, BR(\sigma_1^{\tilde{F}}) \rangle}(I))$. Also, let $\mathcal{G} = \{I \in \mathcal{I}_2^{\tilde{R}} | n(I) \geq o(I)\}$, and $\mathcal{L} = \mathcal{I}_2^{\tilde{R}} \setminus G$. Now consider the dot product of the difference between the two sums

$$
\sum_{I \in \mathcal{I}_2^{\tilde{R}}} w_I (n(I) - o(I))
$$

$$
\begin{aligned}
&= \sum_{I\in\mathcal{L}} w_I(n(I)-o(I)) + \sum_{I\in\mathcal{G}} w_I(n(I)-o(I)) \\
&\leq \sum_{I\in\mathcal{L}} 0(n(I)-o(I)) + \sum_{I\in\mathcal{G}} 1(n(I)-o(I)) \\
&= \sum_{I\in\mathcal{I}_2^{\tilde{R}}} (n(I)-o(I)) - \sum_{I\in\mathcal{L}} (n(I)-o(I)) \\
&\leq \sum_{I\in\mathcal{I}_2^{\tilde{R}}} (n(I)-o(I)) + |\mathcal{L}|\epsilon_S && \text{By B.1,B.2,B.3} \\
&\leq \epsilon_R + |\mathcal{L}|\epsilon_S && \text{By B.4} \qquad \text{(B.5)}
\end{aligned}
$$

There are now two cases. The first case is $\mathcal{G}$ is empty, implying $n(I) < o(I)$ for all I. Therefore $\sum_{I\in\mathcal{I}_2^{\tilde{R}}} w_I n(I) \leq \sum_{I\in\mathcal{I}_2^{\tilde{R}}} w_I o(I)$. This trivially gives us the desired bound. The second case is $\mathcal{G}$ is not empty. Equation B.5 is maximised when $\mathcal{L}$ is as large as possible, which occurs when $|\mathcal{L}| = |\mathcal{I}_2^{\tilde{R}}| - 1$. This gives us the desired bound $\sum_{I\in\mathcal{I}_2^{\tilde{R}}} w_I n(I) \leq (|\mathcal{I}_2^{\tilde{R}}|-1)\epsilon_S + \epsilon_R + \sum_{I\in\mathcal{I}_2^{\tilde{R}}} w_I o(I)$. ■

Theorem 18 *Let σ be a equilibrium profile approximation, with a bound on player 2 counterfactual regret $\max_{a\in(I)} v_2^\sigma(I,a) - \sum_{b\in A(I)} \sigma_2(I,b) v_2^\sigma(I,b) \leq \epsilon_S$ for all I in $\mathcal{I}_2^{\tilde{R}}$. Let $\tilde{\sigma}$ be the re-solved strategy, with a bound ϵ_R on the exploitability in the re-solving game. Then the exploitability of σ is increased by no more than $(|\mathcal{I}_2^{\tilde{R}}|-1)\epsilon_S + \epsilon_R$ if we use $\tilde{\sigma}$ in the subgame:*

$$u_2^{\langle \sigma_{1[S\leftarrow\tilde{\sigma}]}, BR(\sigma_{1[S\leftarrow\tilde{\sigma}]})\rangle} \leq (|\mathcal{I}_2^{\tilde{R}}|-1)\epsilon_S + \epsilon_R + u_2^{\langle \sigma_1, BR(\sigma_1)\rangle}$$

Proof. Let $\hat{\sigma} = \langle \sigma_{1[S\leftarrow\tilde{\sigma}_1]}, BR(\sigma_{1[S\leftarrow\tilde{\sigma}_1]})\rangle$, and Z_S be the set of states in the subgame. In this case,

$$
\begin{aligned}
u_2^{\hat{\sigma}} &= \sum_{z in Z\setminus Z_S} \pi^{\hat{\sigma}}(z) u_2(z) + \sum_{z\in Z_S} \pi^{\hat{\sigma}}(z) u_2(z) \\
&= \sum_{z\in Z\setminus Z_S} \pi^{\langle \sigma_1, BR(\sigma_1)\rangle}(z) u_2(z) + \sum_{z\in Z_S} \pi^{\hat{\sigma}}(z) u_2(z) \qquad \text{(B.6)}
\end{aligned}
$$

Considering only the second sum, rearranging the terms and using Lemma 19

$$
\begin{aligned}
\sum_{z\in Z_S} \pi^{\hat{\sigma}}(z) u_2(z) &= \sum_{I\in\mathcal{I}_2^R} u_2^{\hat{\sigma}}(I) \\
&= \sum_{I\in\mathcal{I}_2^{\tilde{R}}} \pi_2^{\hat{\sigma}}(I) \tilde{u}_2^{\langle \tilde{\sigma_1}, \hat{\sigma_2}^{\tilde{F}}\rangle}(I)
\end{aligned}
$$

A best response must have no less utility than $\hat{\sigma}_2^{\tilde{F}}$, so we can apply Lemma 20

$$\begin{aligned}
&\leq \sum_I \pi_2^{\hat{\sigma}}(I)\tilde{u}_2^{\langle\tilde{\sigma}_1, BR(\tilde{\sigma}_1)\rangle}(I) \\
&\leq (|\mathcal{I}_2^{\tilde{R}}| - 1)\epsilon_S + \epsilon_R + \sum_I \pi_2^{\hat{\sigma}}(I)\tilde{u}_2^{\langle\sigma_1^{\tilde{F}}, BR(\sigma_1^{\tilde{F}})\rangle}(I)
\end{aligned}$$

Because $\tilde{u}_2^{\sigma^{\tilde{F}}}(I) = v_2^{\sigma}(I)$ and $\tilde{u}_2(I \cdot T) = v_2^{\sigma}(I)$ for all I, $BR(\sigma_1^{\tilde{F}})$ can always pick action F, and we can directly use $BR(\sigma_1^{\tilde{F}})$ in the real game, with the same counterfactual value.

$$= (|\mathcal{I}_2^{R}| - 1)\epsilon_S + \epsilon_R + \sum_I \pi_2^{\hat{\sigma}}(I)v_2^{\langle\sigma_1, BR(\sigma_1)\rangle}(I)$$

Putting this back into line B.6, and noting that a best response can only increase the utility, we get

$$\begin{aligned}
u_2^{\hat{\sigma}} &= (|\mathcal{I}_2^{R}| - 1)\epsilon_S + \epsilon_R + u_2^{\langle\sigma_1, \hat{\sigma}_{[S \leftarrow BR(\sigma)}\rangle} \\
&\leq (|\mathcal{I}_2^{\tilde{R}}| - 1)\epsilon_S + \epsilon_R + u_2^{\langle\sigma_1, BR(\sigma_1)\rangle}
\end{aligned}$$

■

Appendix C

DeepStack Proofs

In this section, I present proofs of the arguments given in Section 6.2. The proofs are joint work with Trevor Davis, and were originally published in Science as supplementary material [65]. They use the same terminology as Appendix B, as well as some new terminology introduced below.

The main result is Theorem 12, giving an exploitability bound on the continual re-solving strategy after making d actions. Roughly speaking, the exploitability increases linearly with the estimation quality ϵ_E for each re-solving step: estimation error does not grow multiplicatively over time.

Theorem 12 *Assume we have some initial opponent constraint values w from a solution generated using at least T iterations of CFR-D, we use at least T iterations of CFR-D to solve each re-solving game, and we use a subtree value estimator such that $\min_{\sigma_S^* \in NE_S} \sum_{I \in \mathcal{I}_2^S} |v^{\sigma_S^*}(I) - v_I| \leq \epsilon_E$. Then after d re-solving steps the exploitability of the resulting strategy is no more than $(d+1)k/\sqrt{T} + (2d+1)j\epsilon_E$ for some constants k, j specific to both the game and how it is split into subgames.*

The proof of Theorem 12 requires three parts. First, Lemmas 21 through 25 give a bound on how much re-solving increases the exploitability, if we use constraints that include error from the subgame evaluation function. Second, Lemma 26 gives a bound on exploitability when solving a game using CFR and a subgame evaluation function. Finally, Lemma 27 shows that the optimal-opponent best-response values DeepStack uses for constraints does not increase exploitability. Together, the parts describe one step of continual re-solving.

In games that have many publicly visible actions or chance events, one convenient way to create subgames is to consider public states. A public action is an action (or chance event) a such that for all information sets I, if $\exists h \in I$ where $\exists j \in H, j \cdot a \sqsubseteq h$, then $\forall h \in I$ $\exists j \in H, j \cdot a \sqsubseteq h$. A public state, defined by the sequence of public actions taken in a game, is a set of information sets. Because the actions are public and therefore partition the information sets in a game, a public state is the root of a subgame. A public state and all its descendant states is a subgame.

For example, in HULHE, all of the player actions are public actions: both players know (and remember) the actions made by both players. Similarly, the public board cards are public actions: both players know the public cards, and do not mistake them for other possible public board cards. A sequence of betting and board cards describes a public state consisting of all information sets that only differ in the private cards for the players.

Without loss of generality, we will say player 1 is performing the continual re-solving, and call player 2 the opponent. We write $\mathcal{I}_2^S$ for the collection of player 2 information sets in a subgame S. Let $G\langle S, \sigma, w\rangle$ be the re-solving gadget game, where S is some subgame, σ is used to get player 1 probabilities $\pi_{-2}^{\sigma}(h)$ for each $h \in S$, and c is a vector where c_I gives the value of player 2 taking the terminate action (T) from information set $I \in \mathcal{I}_2^S$. $\mathrm{cvbr}_2^{\sigma}(I)$, defined in Section 2.4.3, gives the player 2 counterfactual value of information set I given they play a counterfactual value best response to our strategy σ. For a subtree strategy σ^S, we write $\sigma \to \sigma^S$ for the strategy that plays according to σ^S for any state in the subtree and according to σ otherwise. For the gadget game $G\langle S, \sigma, c\rangle$, the gadget value of a subtree strategy σ^S is defined to be:

$$\mathrm{GV}_{c,\sigma}^S(\sigma^S) = \sum_{I \in \mathcal{I}_2^S} \max(c_I, \mathrm{cvbr}_2^{\sigma \to \sigma^S}(I)),$$

and the underestimation error (as discussed in Section 5.4.2) is defined to be:

$$\mathrm{U}_{c,\sigma}^S = \min_{\sigma^S} \mathrm{GV}_{c,\sigma}^S(\sigma^S) - \sum_{I \in \mathcal{I}_2^S} c_I.$$

Lemma 21 *The game value of a gadget game $G\langle S, \sigma, c\rangle$ is*

$$\sum_{I \in \mathcal{I}_2^S} c_I + U^S_{c,\sigma}.$$

Proof. Let $\tilde{\sigma}_2^S$ be a gadget game strategy for player 2 which must choose from the F and T actions at starting information set I. Let $\tilde{u}$ be the utility function for the gadget game.

$$\begin{aligned}
\min_{\sigma_1^S} \max_{\tilde{\sigma}_2^S} \tilde{u}(\sigma_1^S, \tilde{\sigma}_2^S) &= \min_{\sigma_1^S} \max_{\sigma_2^S} \sum_{I \in \mathcal{I}_2^S} \frac{\pi_{-2}^{\sigma}(I)}{\sum_{I' \in \mathcal{I}_2^S} \pi_{-2}^{\sigma}(I')} \max_{a \in \{F,T\}} \tilde{u}^{\sigma^S}(I, a) \\
&= \min_{\sigma_1^S} \max_{\sigma_2^S} \sum_{I \in \mathcal{I}_2^S} \max(c_I, \sum_{h \in I} \pi_{-2}^{\sigma}(h) u^{\sigma^S}(h))
\end{aligned}$$

A best response can maximize utility at each information set independently:

$$\begin{aligned}
&= \min_{\sigma_1^S} \sum_{I \in \mathcal{I}_2^S} \max(c_I, \max_{\sigma_2^S} \sum_{h \in I} \pi_{-2}^{\sigma}(h) u^{\sigma^S}(h)) \\
&= \min_{\sigma_1^S} \sum_{I \in \mathcal{I}_2^S} \max(c_I, \text{cvbr}_2^{\sigma \to \sigma_1^S}(I)) \\
&= \text{U}^S_{c,\sigma} + \sum_{I \in \mathcal{I}_2^S} c_I
\end{aligned}$$

■

Lemma 22 *If our strategy σ^S is ϵ-exploitable in the gadget game $G\langle S, \sigma, c\rangle$, then $GV^S_{c,\sigma}(\sigma^S) \leq \sum_{I \in \mathcal{I}_2^S} c_I + U^S_{c,\sigma} + \epsilon$*

Proof. This follows from Lemma 21 and the definitions of ϵ-Nash, $\text{U}^S_{c,\sigma}$, and $\text{GV}^S_{c,\sigma}(\sigma^S)$. ■

Lemma 23 *Given an ϵ_O-exploitable σ in the original game, if we replace a subgame with a strategy σ^S such that $cvbr_2^{\sigma\to\sigma^S}(I) \leq c_I$ for all $I \in \mathcal{I}_2^S$, then the new combined strategy has an exploitability no more than $\epsilon_O + EXP^S_{c,\sigma}$ where*

$$EXP^S_{c,\sigma} = \sum_{I\in\mathcal{I}_2^S} \max(cvbr_2^\sigma(I), c_I) - \sum_{I\in\mathcal{I}_2^S} cvbr_2^\sigma(I)$$

Proof. We only care about the information sets where the opponent's counterfactual value increases, and a worst case upper bound occurs when the opponent best response would reach every such information set with probability 1, and never reach information sets where the value decreased.

Let $Z[S] \subseteq Z$ be the set of terminal states reachable from some $h \in S$ and let v_2 be the game value of the full game for player 2. Let σ_2 be a best response to σ and let σ_2^S be the part of σ_2 that plays in the subtree rooted at S. Then necessarily σ_2^S achieves counterfactual value $\mathrm{cvbr}_2^\sigma(I)$ at each $I \in \mathcal{I}_2^S$.

$$\begin{aligned}
&\max_{\sigma_2^*}(u(\sigma\to\sigma^S, \sigma_2^*))\\
&\quad= \max_{\sigma_2^*}\Bigg[\sum_{z\in Z[S]} \pi_{-2}^{\sigma\to\sigma^S}(z)\pi_2^{\sigma_2^*}(z)u(z) + \sum_{z\in Z\setminus Z[S]} \pi_{-2}^{\sigma\to\sigma^S}(z)\pi_2^{\sigma_2^*}(z)u(z)\Bigg]\\
&\quad= \max_{\sigma_2^*}\Bigg[\sum_{z\in Z[S]} \pi_{-2}^{\sigma\to\sigma^S}(z)\pi_2^{\sigma_2^*}(z)u(z) - \sum_{z\in Z[S]} \pi_{-2}^{\sigma}(z)\pi_2^{\sigma_2^*\to\sigma_2^S}(z)u(z)\\
&\qquad\quad + \sum_{z\in Z[S]} \pi_{-2}^{\sigma}(z)\pi_2^{\sigma_2^*\to\sigma_2^S}(z)u(z) + \sum_{z\in Z\setminus Z[S]} \pi_{-2}^{\sigma}(z)\pi_2^{\sigma_2^*}(z)u(z)\Bigg]\\
&\quad\leq \max_{\sigma_2^*}\Bigg[\sum_{z\in Z[S]} \pi_{-2}^{\sigma\to\sigma^S}(z)\pi_2^{\sigma_2^*}(z)u(z) - \sum_{z\in Z[S]} \pi_{-2}^{\sigma}(z)\pi_2^{\sigma_2^*\to\sigma_2^S}(z)u(z)\Bigg]\\
&\qquad\quad + \max_{\sigma_2^*}\Bigg[\sum_{z\in Z[S]} \pi_{-2}^{\sigma}(z)\pi_2^{\sigma_2^*\to\sigma_2^S}(z)u(z) + \sum_{z\in Z\setminus Z[S]} \pi_{-2}^{\sigma}(z)\pi_2^{\sigma_2^*}(z)u(z)\Bigg]\\
&\quad\leq \max_{\sigma_2^*}\Bigg[\sum_{I\in\mathcal{I}_2^S}\sum_{h\in I} \pi_{-2}^{\sigma}(h)\pi_2^{\sigma_2^*}(h)u^{\sigma^S,\sigma_2^*}(h)\\
&\qquad\quad - \sum_{I\in\mathcal{I}_2^S}\sum_{h\in I} \pi_{-2}^{\sigma}(h)\pi_2^{\sigma_2^*}(h)u^{\sigma,\sigma_2^S}(h)\Bigg] + \max_{\sigma_2^*}(u(\sigma,\sigma_2^*))
\end{aligned}$$

By perfect recall $\pi_2(h) = \pi_2(I)$ for each $h \in I$:

$$\begin{aligned}
&\leq \max_{\sigma_2^*}\Bigg[\sum_{I\in\mathcal{I}_2^S} \pi_2^{\sigma_2^*}(I)\Bigg(\sum_{h\in I}\pi_{-2}^{\sigma}(h)u^{\sigma^S,\sigma_2^*}(h) - \sum_{h\in I}\pi_{-2}^{\sigma}(h)u^{\sigma,\sigma_2^S}(h)\Bigg)\Bigg] \\
&\qquad + v_2 + \epsilon_O \\
&= \max_{\sigma_2^*}\Bigg[\sum_{I\in\mathcal{I}_2^S} \pi_2^{\sigma_2^*}(I)\pi_{-2}^{\sigma}(I)\Bigg(\mathrm{cvbr}_2^{\sigma\to\sigma^S}(I) - \mathrm{cvbr}_2^{\sigma}(I)\Bigg)\Bigg] + v_2 + \epsilon_O \\
&\leq \qquad \Bigg[\sum_{I\in\mathcal{I}_2^S} \max(\mathrm{cvbr}_2^{\sigma\to\sigma^S}(I) - \mathrm{cvbr}_2^{\sigma}(I), 0)\Bigg] + v_2 + \epsilon_O \\
&\leq \qquad \Bigg[\sum_{I\in\mathcal{I}_2^S} \max(c_I - \mathrm{cvbr}_2^{\sigma}(I), \mathrm{cvbr}_2^{\sigma}(I) - \mathrm{cvbr}_2^{\sigma}(I))\Bigg] + v_2 + \epsilon_O \\
&= \qquad \Bigg[\sum_{I\in\mathcal{I}_2^S} \max(\mathrm{cvbr}_2^{\sigma}(I), c_I) - \sum_{I\in\mathcal{I}_2^S}\mathrm{cvbr}_2^{\sigma}(I)\Bigg] + v_2 + \epsilon_O
\end{aligned}$$

■

Lemma 24 *Given an ϵ_O-exploitable σ in the original game, if we replace the strategy in a subgame with a strategy σ^S that is ϵ_S-exploitable in the gadget game $G\langle S,\sigma,c\rangle$, then the new combined strategy has an exploitability no more than $\epsilon_O + EXP_{c,\sigma}^S + U_{c,\sigma}^S + \epsilon_S$.*

Proof. We use that $\max(a,b) = a + b - \min(a,b)$. From applying Lemma 23 with $c_I = \mathrm{cvbr}_2^{\sigma\to\sigma^S}(I)$ and expanding $\mathrm{EXP}^S_{\mathrm{cvbr}_2^{\sigma\to\sigma^S},\sigma}$ we get exploitability no more than $\epsilon_O - \sum_{I\in\mathcal{I}_2^S}\mathrm{cvbr}_2^{\sigma}(I)$ plus

$$\begin{aligned}
&\sum_{I\in\mathcal{I}_2^S} \max(\mathrm{cvbr}_2^{\sigma\to\sigma^S}(I), \mathrm{cvbr}_2^{\sigma(I)} \\
&\quad \leq \sum_{I\in\mathcal{I}_2^S} \max(\mathrm{cvbr}_2^{\sigma\to\sigma^S}(I), \max(c_I, \mathrm{cvbr}_2^{\sigma}(I)) \\
&\quad = \sum_{I\in\mathcal{I}_2^S} \big(\mathrm{cvbr}_2^{\sigma\to\sigma^S}(I) + \max(c_I, \mathrm{cvbr}_2^{\sigma}(I)) \\
&\qquad\quad - \min(\mathrm{cvbr}_2^{\sigma\to\sigma^S}(I), \max(c_I, \mathrm{cvbr}_2^{\sigma}(I)))\big) \\
&\quad \leq \sum_{I\in\mathcal{I}_2^S} \big(\mathrm{cvbr}_2^{\sigma\to\sigma^S}(I) + \max(c_I, \mathrm{cvbr}_2\sigma(I)) \\
&\qquad\quad - \min(\mathrm{cvbr}_2^{\sigma\to\sigma^S}(I), c_I)\big) \\
&\quad = \sum_{I\in\mathcal{I}_2^S} \big(\max(c_I, \mathrm{cvbr}_2^{\sigma}(I)) + \max(c_I, \mathrm{cvbr}_2^{\sigma\to\sigma^S}(I)) - c_I\big)
\end{aligned}$$

$$= \sum_{I \in \mathcal{I}_2^S} \max(c_I, \mathrm{cvbr}_2^\sigma(I)) + \sum_{I \in \mathcal{I}_2^S} \max(c_I, \mathrm{cvbr}_2^{\sigma \to \sigma^S}(I)) - \sum_{I \in \mathcal{I}_2^S} c_I$$

From Lemma 22 we get

$$\leq \sum_{I \in \mathcal{I}_2^S} \max(c_I, \mathrm{cvbr}_2^\sigma(I)) + \mathrm{U}_{c,\sigma}^S + \epsilon_S$$

Adding $\epsilon_O - \sum_I \mathrm{cvbr}_2^\sigma(I)$ we get the upper bound $\epsilon_O + \mathrm{EXP}_{c,\sigma}^S + \mathrm{U}_{c,\sigma}^S + \epsilon_S$. ■

Lemma 25 *Assume we are performing one step of re-solving on subtree S, with constraint values c approximating opponent best-response values to the previous strategy σ, with an approximation error bound $\sum_I |c_I - cvbr_2^\sigma(I)| \leq \epsilon_E$. Then we have $EXP_{c,\sigma}^S + U_{c,\sigma}^S \leq \epsilon_E$.*

Proof. $\mathrm{EXP}_{c,\sigma}^S$ measures the amount that the c_I exceed $\mathrm{cvbr}_2^\sigma(I)$, while $\mathrm{U}_{c,\sigma}^S$ bounds the amount that the c_I underestimate $\mathrm{cvbr}_2^{\sigma \to \sigma^S}(I)$ for any σ^S, including the original σ. Thus, together they are bounded by $|c_I - \mathrm{cvbr}_2^\sigma(I)|$:

$$\begin{aligned}
\mathrm{EXP}_{c,\sigma}^S + \mathrm{U}_{c,\sigma}^S &= \sum_{I \in \mathcal{I}_2^S} \max(\mathrm{cvbr}_2^\sigma(I), c_I) - \sum_{I \in \mathcal{I}_2^S} \mathrm{cvbr}_2^\sigma(I) \\
&\quad + \min_{\sigma^S} \sum_{I \in \mathcal{I}_2^S} \max(c_I, \mathrm{cvbr}_2^{\sigma \to \sigma^S}(I)) - \sum_{I \in \mathcal{I}_2^S} c_I \\
&\leq \sum_{I \in \mathcal{I}_2^S} \max(\mathrm{cvbr}_2^\sigma(I), c_I) - \sum_{I \in \mathcal{I}_2^S} \mathrm{cvbr}_2^\sigma(I) \\
&\quad + \sum_{I \in \mathcal{I}_2^S} \max(c_I, \mathrm{cvbr}_2^\sigma(I)) - \sum_{I \in \mathcal{I}_2^S} c_I \\
&= \sum_{I \in \mathcal{I}_2^S} [\max(c_I - \mathrm{cvbr}_2^\sigma(I), 0) + \max(\mathrm{cvbr}_2^\sigma(I) - c_I, 0)] \\
&= \sum_{I \in \mathcal{I}_2^S} |c_I - \mathrm{cvbr}_2^\sigma(I)| \leq \epsilon_E
\end{aligned}$$

■

Lemma 26 *Assume we are solving a game G with T iterations of CFR-D where for both players p, subtrees S, and times t, we use subtree values v_I for all information sets I at the root of S from some suboptimal black box estimator. If the estimation error is bounded, so that* $\min_{\sigma^*_S \in NE_S} \sum_{I \in \mathcal{I}^S_2} |v^{\sigma^*_S}(I) - v_I| \leq \epsilon_E$, *then the trunk exploitability is bounded by* $k_G/\sqrt{T} + j_G\epsilon_E$ *for some game specific constant* $k_G, j_G \geq 1$ *which depend on how the game is split into a trunk and subgames.*

Proof. This follows from a modified version the proof of Theorem 2 of Burch et al. [15], which uses a fixed error ϵ and argues by induction on information sets. Instead, we argue by induction on entire public states.

For every public state s, let N_s be the number of subgames reachable from s, including any subgame rooted at s. Let $Succ(s)$ be the set of our public states which are reachable from s without going through another of our public states on the way. Note that if s is in the trunk, then every $s' \in Succ(s)$ is in the trunk or is the root of a subgame. Let $D_{TR}(s)$ be the set of our trunk public states reachable from s, including s if s is in the trunk. We argue that for any public state s where we act in the trunk or at the root of a subgame

$$\sum_{I \in s} R^T_{full}(I)^+ \leq \sum_{s' \in D_{TR}(s)} \sum_{I \in s'} R^T(I)^+ + TN_s\epsilon_E \tag{C.1}$$

First note that if no subgame is reachable from s, then $N_s = 0$ and the statement follows from Lemma 7 of [94]. For public states from which a subgame is reachable, we argue by induction on $|D_{TR}(s)|$.

For the base case, if $|D_{TR}(s)| = 0$ then s is the root of a subgame S, and by assumption there is a Nash Equilibrium subgame strategy σ^*_S that has regret no more than ϵ_E. If we implicitly play σ^*_S on each iteration of CFR-D, we thus accrue $\sum_{I \in s} R^T_{full}(I)^+ \leq T\epsilon_E$.

For the inductive hypothesis, we assume that (C.1) holds for all s such that $|D_{TR}(s)| < k$.

Consider a public state s where $|D_{TR}(s)| = k$. By Lemma 5 of [94] we have

$$\sum_{I \in s} R^T_{full}(I)^+ \leq \sum_{I \in s} \left[R^T(I) + \sum_{I' \in Succ(I)} R^T_{full}(I)^+ \right]$$

$$= \sum_{I \in s} R^T(I) + \sum_{s' \in Succ(s)} \sum_{I' \in s'} R^T_{full}(I')^+$$

For each $s' \in Succ(s)$, $D(s') \subset D(s)$ and $s \notin D(s')$, so $|D(s')| < |D(s)|$ and we can apply the inductive hypothesis to show

$$\begin{aligned}
\sum_{I \in s} R^T_{full}(I)^+ &\le \sum_{I \in s} R^T(I) + \sum_{s' \in Succ(s)} \left[\sum_{s'' \in D(s')} \sum_{I \in s''} R^T(I)^+ + T N_{s'} \epsilon_E \right] \\
&\le \sum_{s' \in D(s)} \sum_{I \in s'} R^T(I)^+ + T\epsilon_E \sum_{s' \in Succ(s)} N_{s'} \\
&= \sum_{s' \in D(s)} \sum_{I \in s'} R^T(I)^+ + T\epsilon_E N_s
\end{aligned}$$

This completes the inductive argument. By using regret matching in the trunk, we ensure $R^T(I) \le \Delta\sqrt{AT}$, proving the lemma for $k_G = \Delta|\mathcal{I}_{\mathcal{TR}}|\sqrt{A}$ and $j_G = N_{root}$. ■

Lemma 27 *Given our strategy σ, if the opponent is acting at the root of a subgame S from a set of actions A, with opponent best-response values $cvbr^{\sigma}_2(I \cdot a)$ after each action $a \in A$, then replacing our subtree strategy with any strategy that satisfies the opponent constraints $c_I = \max_{a \in A} cvbr^{\sigma}_2(I \cdot a)$ does not increase our exploitability.*

Proof. If the opponent is playing a best response, every counterfactual value c_I before the action must either satisfy $c_I = \text{cvbr}^{\sigma}_2(I) = \max_{a \in A} \text{cvbr}^{\sigma}_2(I \cdot a)$, or not reach state s with private information I. If we replace our strategy in S with a strategy σ'_S such that $\text{cvbr}^{\sigma'_S}_2(I \cdot a) \le \text{cvbr}^{\sigma}_2(I)$ we preserve the property that $\text{cvbr}^{\sigma'}_2(I) = \text{cvbr}^{\sigma}_2(I)$. ■

Theorem 12 *Assume we have some initial opponent constraint values w from a solution generated using at least T iterations of CFR-D, we use at least T iterations of CFR-D to solve each re-solving game, and we use a subtree value estimator such that $\min_{\sigma_S^* \in NE_S} \sum_{I \in \mathcal{I}_2^S} |v^{\sigma_S^*}(I) - v_I| \leq \epsilon_E$. Then after d re-solving steps the exploitability of the resulting strategy is no more than $(d+1)k/\sqrt{T} + (2d+1)j\epsilon_E$ for some constants k, j specific to both the game and how it is split into subgames.*

Proof. Continual re-solving begins by solving from the root of the entire game, which we label as subtree S_0. We use CFR-D with the value estimator in place of subgame solving in order to generate an initial strategy σ_0 for playing in S_0. By Lemma 26, the exploitability of σ_0 is no more than $k_0/\sqrt{T} + j_0\epsilon_E$.

For each step of continual re-solving $i = 1, ..., d$, we are re-solving some subtree S_i. From the previous step of re-solving, we have approximate opponent best-response counterfactual values $\widetilde{\text{cvbr}}_2^{\sigma_{i-1}}(I)$ for each $I \in \mathcal{I}_2^{S_{i-1}}$, which by the estimator bound satisfy $|\sum_{I \in \mathcal{I}_2^{S_{i-1}}}(\text{cvbr}_2^{\sigma_{i-1}}(I) - \widetilde{\text{cvbr}}_2^{\sigma_{i-1}}(I))| \leq \epsilon_E$. Updating these values at each public state between S_{i-1} and S_i as described in the paper yields approximate values $\widetilde{\text{cvbr}}_2^{\sigma_{i-1}}(I)$ for each $I \in \mathcal{I}_2^{S_i}$ which by Lemma 27 can be used as constraints $c_{I,i}$ in re-solving. Lemma 25 with these constraints gives us the bound $\text{EXP}_{c_i,\sigma_{i-1}}^{S_i} + \text{U}_{c_i,\sigma_{i-1}}^{S_i} \leq \epsilon_E$. Thus by Lemma 24 and Lemma 26 we can say that the increase in exploitability from σ_{i-1} to σ_i is no more than $\epsilon_E + \epsilon_{S_i} \leq \epsilon_E + k_i/\sqrt{T} + j_i\epsilon_E \leq k_i/\sqrt{T} + 2j_i\epsilon_E$.

Let $k = \max_i k_i$ and $j = \max_i j_i$. Then after d re-solving steps, the exploitability is bounded by $(d+1)k/\sqrt{T} + (2d+1)j\epsilon_E$. ■