Toward Generality: Building Better Counterfactual Regret Minimization for Imperfect Information Games

Toward Generality: Building Better Counterfactual Regret Minimization for Imperfect Information Games

a thesis presented
by
Mason G. Meyer
to
The Department of Computer Science

in partial fulfillment of the requirements
for the degree of
Bachelor of Arts (Honors)
in the subject of
Computer Science

Harvard University
Cambridge, Massachusetts
November 2021

Advised by David Parkes and Hugh Zhang.

I'm extremely grateful for the help and support of both.

Abstract

The imperfect information game is a mathematical model that is very useful for modeling interactions in a wide variety of domains, and algorithms that compute the solutions of such games are extremely valuable. Counterfactual regret minimization (CFR) is a very promising algorithmic paradigm for this setting. However, current high-performance CFR algorithms are designed to work well for only a limited subset of games with a small number of available actions, such as poker, and thus have narrow applicability. This thesis discusses the importance of developing more general algorithms to solve mathematical games, justifies why CFR is a promising path forward for creating these solutions, and ultimately addresses this situation by proposing two novel variants of CFR, one of which in particular is especially promising as a general solution. By approximating the value of information to be gained at different times and states of the game, these algorithms match the performance of current best methods on small games while also, for one algorithm, extending to large games by performing faster than current high-performance methods by an exponential factor, serving as an example of a more general solution for imperfect information games.

Contents

1 Introduction

1.1 Motivation

Modern machine learning and statistical models have the capacity to aid humans in a wide variety of decision-making tasks, and indeed, many such models are widely used today, often in high-stakes environments that range from medicine to criminal law.

While a wide variety of different models optimized for different features exist, one class of models that is particularly interesting is games. A mathematical game is a type of model that is uniquely positioned to represent interactions between different agents, and using games as models of different features of the world has been critical to develop human understanding of systems as disparate as global economy and biological evolution.

Historically, there has been a large amount of philosophical interest in a feature of games referred to as *imperfect information*, where participants in such a system are able or forced to hide information from one another. However, algorithmic game theory, the field dedicated to using computational tools to solve and better understand games, has largely focused on games where there is no hidden information.

The modern world is currently facing an ever-increasing onslaught of applications that require a sophisticated knowledge of imperfect information settings to address properly. Increasingly complicated models of pricing and cybersecurity are two technical areas in which deci-

sions need to be made optimally despite enormous amounts of unknown information. However, these models function equally well in human-centered applications as well, from modeling negotiations to situations of bias.

In recent years, the field of algorithmic game theory has responded to this growing need for good models of hidden information by developing a variety of algorithms precisely for imperfect information games. However, because hidden information introduces so much complexity into models, many of these algorithms are optimized only for a narrowly-defined subset of games. That is, they can be said to lack *generality*.

1.2 Role of this Thesis

The purpose of this thesis is to offer novel solutions and progress on the matter of how to best increase the generality of how equilibria are computed in imperfect-information zero-sum games. These terms, if unfamiliar, will be defined shortly in the coming pages.

1.2.1 The Path Forward

To move toward this goal, in this thesis, I will first provide an overview of games and how they are solved, introducing the requisite background while also proposing an algorithmic paradigm known as *counterfactual regret minimization* (CFR) as the best path toward generality. I will explore this paradigm in depth, paying special attention to its limitations, and highlighting how other researchers have overcome these barriers in recent years through tools such as Monte Carlo methods and function-approximating neural networks.

After establishing the current obstacles preventing CFR from serving as a general solution, in Chapter 5 I will propose a two new variants within this algorithmic paradigm directly designed to be general solu-

tions that operate by approximating the value of information at different game states and times. I will experimentally test these variants against current benchmarks to show not only improved practical performance, but also significant theoretical improvements (for one variant in particular) by performing faster than current best methods by an exponential factor. This allows for applications to much larger and differently structured games without narrow optimization, dramatically increasing the generality of the CFR technique.

1.2.2 Thesis Contribution

The primary contribution of my thesis consists of the two novel classes of algorithms and their applications to large games that cannot be addressed by current methods. These variants are not intended to serve as "magic bullets" that solve the generality problem completely. Rather, they serve to introduce a framework based on approximating the value of information that has promising potential for future use. The framework can guide the design of more general solutions that extend even beyond the CFR variant algorithms I propose, and in Chapter 6 I explore a selection of these applications.

While these novel algorithms and their framework for approaching generality form one possible path toward more general solutions, I also hope that this work highlights the importance of generality in modern algorithmic game theory as a whole, facilitating the creation of models that can help us understand important features of the modern world in a wide variety of settings.

2 Background: Games and How to Solve Them

2.1 Games

What is a game? What does it mean to play a game perfectly? Is this possible for *any* game? As discussed in the introduction, games are mathematical models critical for understanding a variety of systems involving interactions between agents, and the purpose of this thesis is to reveal how we can construct algorithms to solve such games in a more general way. In this chapter, I will introduce some concepts fundamental to game theory that are necessary to better understand the path towards achieving generality in solving these games.

The study of games as mathematical objects dates back centuries, but was largely codified in modern study by John von Neumann (von Neumann 1944). In the framework that von Neumann proposes, a *game* is an object that models strategic actions among rational, self-interested agents by allowing them to perform *actions* at certain points in time that change the state of the game. Players will generally choose the appropriate action from a constrained set of actions according to a *strategy*, which they can change at any point. There are additionally different types of strategies a player can employ: a *pure strategy* is defined by the player deterministically choosing one of the possible actions, while a

mixed strategy is defined by a probability distribution over the different available actions.

In classical game theory, a game has a *solution*, which is the optimal strategy for a player to use in that particular game. For many types of games, this solution is called the *Nash equilibrium*, which is the stable-state description of the strategies of the game's players at the point when each player is aware of the strategies of all other players, and no player has any incentive to modify their own strategy (Nash 1951).

There are a wide variety of different classifications of games, and different classifications have different mathematical properties. In *zero-sum* games, one player's gain is necessarily another player's loss, and there is no change in total utility after the game is over. For example, in most varieties of poker, one player gaining a sum of money is balanced by the other players losing that exact sum amongst themselves. In contrast, a *general-sum* game has no restraints on net change in utility.

Another important classification of games is that of *perfect information* vs *imperfect information* games. In a *perfect information* game, every player has full and equal knowledge of the game state (e.g. tic-tac-toe). In an *imperfect information* game, each player has limited knowledge of the game state (e.g. a card game where you are unable to see the cards of other players). In this thesis, I will be focusing on zero-sum imperfect information games. I will also limit the discussion to games with *perfect recall*, in which players never forget information.

This conceptual breakdown and categorization of games is useful, and makes it apparent that games can be used to model a wide variety of processes. Yes, a parlor game like chess or poker can be modelled mathematically as a game, but so can economic systems, negotiations, and any system in which agents are self-interested and interacting with other actors.

To interact with games in a rigorous way, it is necessary to formalize this definition of a game. Additionally, for algorithms to learn to play games effectively, this formalization allows algorithms to understand game structure. Many different formalizations exist for different classes of games, and there are many different methods to categorize games as well. One particular category of games is known as **normal-form games**.

2.1.1 Normal-form Games

Normal-form games are among the simplest to define, and can model interactions between a finite number of players. In a normal-form game, each player will choose an action available to them, then all actions will be revealed simultaneously. After the actions are revealed, the game is over and each player receives a reward that depends both on their individual action, and the actions of all other players.

A normal-form game can be represented by the tuple: $\langle \mathcal{N}, \mathcal{A}, \boldsymbol{u} \rangle$

- $\mathcal{N}$ is a finite set of n elements, where n is the number of players: $\{1, 2, 3, ..., n\}$.
- $\mathcal{A}$ is the set of all possible action combinations, where each element of $\mathcal{A}$ is a vector of length n. Each vector represents the actions taken by each player. For example, let A_i be the set of actions available to player i. Suppose player 1 chooses action $a_1 \in A_1$, player 2 chooses action $a_2 \in A_2$, and so on. Then $[a_1, a_2, ..., a_n] \in \mathcal{A}$.
- $\boldsymbol{u}$ is the *utility function* that maps each possible element of $\mathcal{A}$ to a vector representing the reward that each player receives. Letting reward be a real number, we have: $\boldsymbol{u} \colon \mathcal{A} \to \mathbb{R}^n$.

We can note that a game is *zero-sum* when the sum of all elements in the vector $\boldsymbol{u}(A)$ for all $A \in \mathcal{A}$ is zero.

To make this model concrete, consider a game of rock-paper-scissors between two people, where each player can choose between the actions R, P, S. Let a reward of -1 correspond to losing, a reward of 1 correspond to winning, and a reward of 0 correspond to a tie. The Nash equilibrium of this game is a *mixed strategy* where the probabilities of selecting each action R, P, S are equal at $\frac{1}{3}$. This game can be modelled as normal-form:

$\mathcal{N} = \{1, 2\}$

(There are two players.)

$\mathcal{A} = \Big\{[R,R],[R,P],[R,S],[P,R],[P,P],[P,S],[S,R],[S,P],[S,S]\Big\}$

($A_1 = A_2 = \{R, P, S\}$, so each possible action combination is represented here.)

$\boldsymbol{u}$:

$[R,R] \mapsto [0,0]$

$[R,P] \mapsto [-1,1]$

$[R,S] \mapsto [1,-1]$

$[P,R] \mapsto [1,-1]$

$[P,P] \mapsto [0,0]$

$[P,S] \mapsto [-1,1]$

$[S,R] \mapsto [-1,1]$

$[S,P] \mapsto [1,-1]$

$[S,S] \mapsto [0,0]$

(R beats S, S beats P, P beats R)

Normal-form games allow us to model a great number of different behaviors, but they still have some limitations. For example, many interesting games involve players taking more than just one action, and these

action may be taken in sequence (e.g. chess, an auction, or any negotiation). Because normal-form games model only one action per player simultaneously, modelling such sequential games as normal-form requires creating a separate action for each possible action sequence. That is, if one wants to represent a multi-action game as a normal-form game, there must be a distinct action representation not only for each possible action sequence for a player, but also for each possible response to any action by any other player.

This large number of elements in $\mathcal{A}$, growing exponentially as the number of turns increases, can make it impractical for algorithms to develop good strategies for games represented in this way. Furthermore, because the entire action sequence is encoded in a single action, it can be very difficult to reason about the state of the game in an intermediary state.

Because of these limitations, the mathematical model that I will use in this thesis to model games is that of the **extensive-form game**.

2.1.2 Extensive-form Games

Extensive-form games, proposed in 1953 by Harold Kuhn (Kuhn and Tucker 1953), model sequential actions between players, where the actions each player takes may depend on the history of previous actions. Extensive form games can be thought of as an extension of normal-form games, though they also represent a wide variety of different *game states* based on the previous actions. These games states and certain properties they possess are organized into a *game tree*.

An extensive-form game can be represented by the tuple: $\langle \mathcal{N}, \mathcal{T}, \mathcal{A}, \boldsymbol{u} \rangle$. The definitions of these terms vary slightly from their definitions for normal-form games.

- $\mathcal{N}$ is a finite set of $n+1$ elements, where n is the number of rational players: $\{1, 2, 3, ..., n, c\}$. The c player is referred to as "chance",

and is used to model probabilistic events in game play, such as a random dice roll.

- $\mathcal{T}$ is the *game tree*. Nodes in this tree represent the game state, while edges represent actions, which move the game to a new state. The game tree contains a variety of pieces of information:
  - $\mathcal{H}$ is a finite set of *histories*. A history is a sequence of actions starting from the root of the game tree. Each game state is modelled by one history.
    - $\mathcal{Z} \subseteq \mathcal{H}$ is the set of *terminal histories*, game states that represent the end of the game.
  - $\boldsymbol{p}$ is a function mapping from a history to the player who takes an action at that game state. $\boldsymbol{p}$: $h \mapsto i \mid h \in \mathcal{H}, i \in \mathcal{N}$.
  - τ is a function that, given a chance node as an input, will return the probability distribution of actions for the chance player at that game state.
- $\mathcal{A}$ is the set of all possible actions performable by any player at any point in the game tree. $\mathcal{A}(h)$ is the set of actions performable at history h by player $\boldsymbol{p}(h)$.
- $\boldsymbol{u}$ is the *utility function* that maps each terminal history of the game tree to a vector representing the reward that each player receives. Letting reward be a real number, we have: $\boldsymbol{u}$: $\mathcal{Z} \to \mathbb{R}^n$.

Understanding this formulation of extensive-form games can be difficult (why, for example, must there be a chance player?). To better illustrate the formulation of extensive form games, let us consider a model game that we will revisit throughout this thesis: Kuhn Poker.

2.2 A Model Game: Kuhn Poker

As a favorite toy game of the imperfect-information game community, Kuhn poker is a wonderfully appropriate place to begin our discussion on imperfect-information games. An understanding of this model game is important because 1) it allows us to better understand the properties of extensive-form games with a concrete example, and 2) Kuhn Poker and similar games will be especially relevant for testing the effectiveness of the novel algorithms I will propose later in this thesis.

Kuhn poker is a simple *imperfect-information game* created by the mathematician Harold W. Kuhn in 1953 (Kuhn and Tucker 1953). It is modelled after poker, and is played by two players who win or lose money that they bet. Reward is measured in units of "chips", and the game is especially useful because it has a known Nash Equilibrium in which the second player will win an average of $\frac{1}{18}$ of a chip per game.

To play the game, two players each *ante* 1 chip into the *pot*. The pot is a pool of chips to be claimed by the winner of the game, and an ante is a forced bet. Each player is then dealt one of three cards, labeled 0, 1, and 2, randomly, and this information is not shared with the other player. Player 1 begins by selecting one of two actions: *pass* or *bet*. Passing moves play to the other player. Betting requires placing one additional chip in the pot. The players alternate performing an action until one of the following occurs: 1) two passes or two bets in a row, or 2) a pass occurs after a bet. If the game ends with situation 1 (two passes or two bets), the player with the higher-numbered card wins the entire pot. If the game ends with situation 2 (pass after bet), the player who made the previous bet wins the entire pot. A summary of the possible terminal action sequences with associated rewards are listed in Table 2.1.

Kuhn poker can very intuitively be modeled as an extensive-form game, which is an important step in allowing algorithms to learn strate-

Terminal Action Sequence	Reward
P1: bet, P2: bet	+2 to high card player, -2 to low card player
P1: bet, P2: pass	+1 to P1, -1 to P2
P1: pass, P2: pass	+1 to high card player, -1 to low card player
P1: pass, P2: bet, P1: bet	+2 to high card player, -2 to low card player
P1: pass, P2: bet, P1: pass	+1 to P2, -1 to P1

Table 2.1: The terminal action sequences of Kuhn Poker with associated rewards

gies for Kuhn poker. Let us reconsider the tuple that defines an extensive-from game, $\langle \mathcal{N}, \mathcal{T}, \mathcal{A}, \boldsymbol{u} \rangle$, defining each of these terms for Kuhn poker.

$\mathcal{N} = \{1, 2, c\}$. There are two players, and the chance player controls the random actions of dealing cards.

The game tree $\mathcal{T}$ can be seen in Figure 2.1. This tree details each possible history, represented as a node. A filled black circle indicates a history h where $\boldsymbol{p}(h) = c$ (the active player is chance). Similarly, a blue square indicates $\boldsymbol{p}(h) = 1$ (Player 1 is acting), and a red circle indicates $\boldsymbol{p}(h) = 2$ (Player 2 is acting). Black diamonds represent terminal histories ($\mathcal{Z}$), and are filled with the reward for player 1.

Continuing with defining the the parameters of our extensive-form representation, $\mathcal{A} = \{\text{Deal:0, Deal:1, Deal:2, Pass, Bet}\}$. For any non-terminal history h such that $\boldsymbol{p}(h) \in \{1, 2\}$, $\mathcal{A}(h) = \{\text{Pass, Bet}\}$.

Lastly, the function $\boldsymbol{u}$ maps a terminal history to a vector of length 2, and is partially defined in Figure 2.1. The reward for Player 1 is listed in the tree, and because Kuhn Poker is zero-sum, the reward for Player 2 is the additive inverse of this number. For example, letting z be the leftmost terminal history in the tree, $\boldsymbol{u}(z) = [-1, +1]$.

Again, understanding the mathematical representations of these games is important because algorithms learning optimal solutions only have access to the information contained in the representation used.

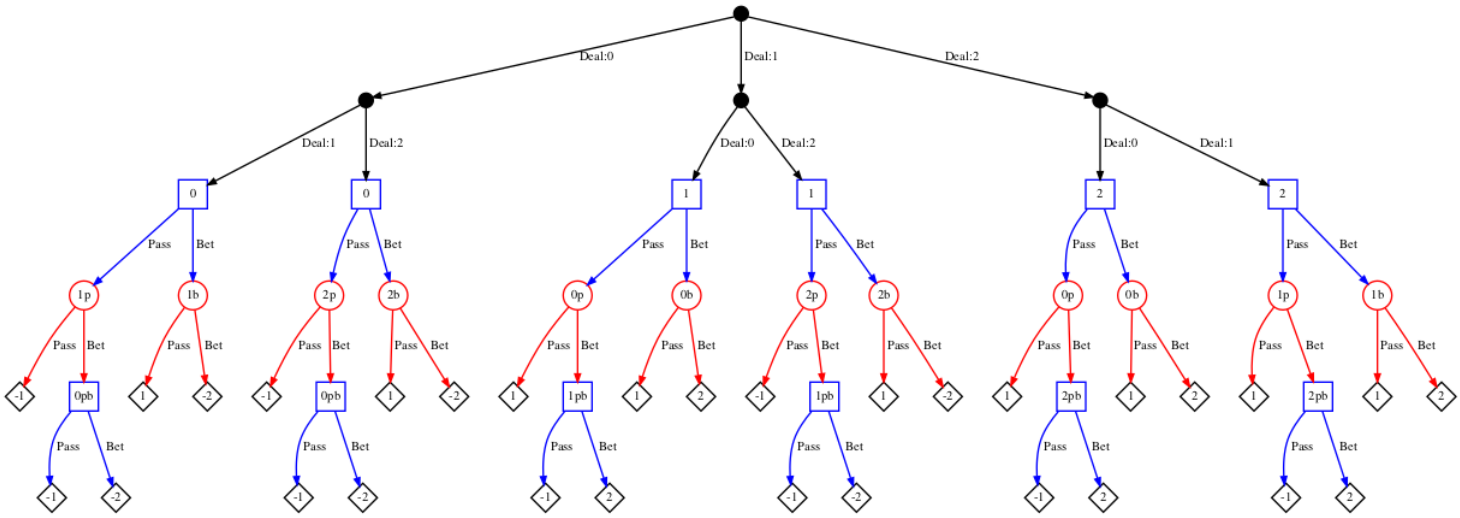


Figure 2.1: The game tree of Kuhn poker. PyGraphViz library used to visualize (Hagberg et al. 2021).

Looking at Figure 2.1, notice how this tree demonstrates that chance actions are considered a part of each history. Because there are $nPr(3,2) = \frac{3!}{(3-2)!} = 6$ ways the cards can be distributed by chance, there are 6 times more terminal histories in Figure 2.1 than terminal action sequences in Table 2.1. If there are a large number of outcomes from chance events, this can dramatically increase the total number of histories. Imagine playing Kuhn poker with a 100-card deck. In this variation, there would be 9900 times more terminal histories than terminal action sequences (of which there are still only 5). Fortunately, extensive-form games offer a simplification via *information sets*.

2.2.1 Imperfect Information

Kuhn poker is an imperfect information game; the game is only partially observable because each player cannot see the other's cards. Because this hidden information cannot be used to inform a player's actions, when multiple histories have the exact same information available to the active player, these histories can be grouped into an *information set*.

To emphasize, information sets are grouped from the information-awareness of the current active player. Information sets are powerful, as the probability distribution over actions for any strategy will be identical for every history in a given information set. For example, when playing Kuhn poker, if Player 1 is given the card labeled 1, the two histories in which Player 2 is given the card 0 and the card 2 are contained within the same information set, and Player 1 has the same probability of betting regardless of which history is the current game state.

Taking a look at Figure 2.1, is it clear how the information sets should be grouped for Kuhn poker? Are all 12 information sets identifiable? Figure 2.2 provides a visualization of the game tree with information sets included. In this figure, dashed boxes group histories into information sets. Note how at each successive layer of the tree, the active player changes, the so the information available to create each information set alternates between players.

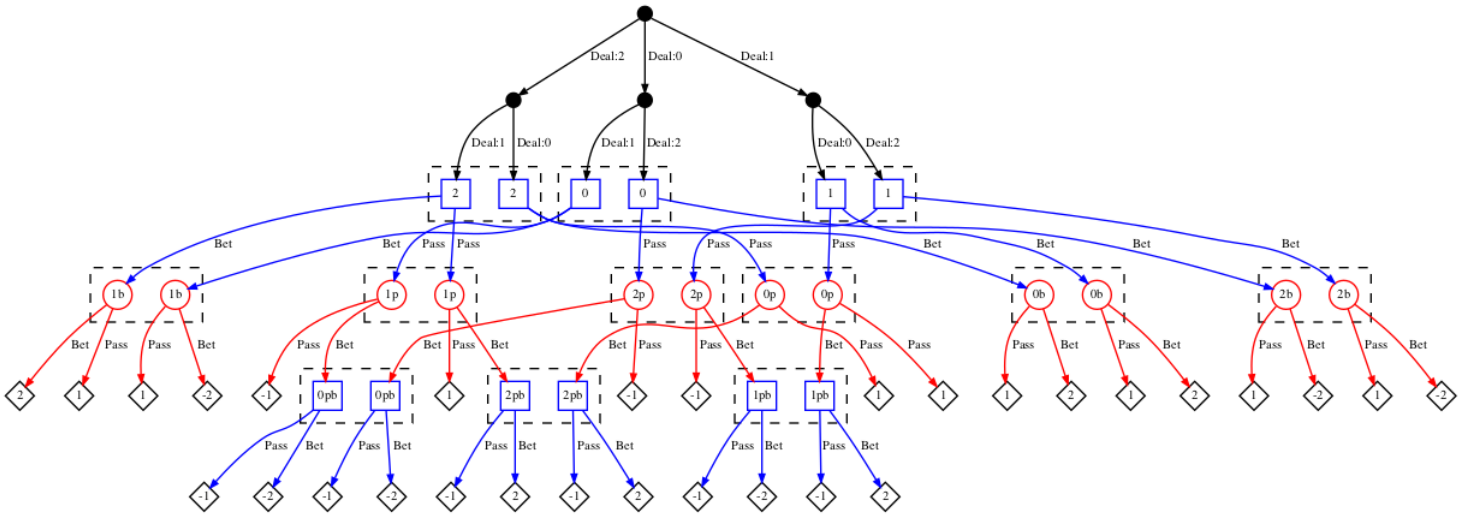


Figure 2.2: The game tree of Kuhn poker, with information sets. PyGraphViz library used to visualize (Hagberg et al. 2021).

At this point, we have a clear model for how to represent games, and an understanding of different tools used to model important features

of imperfect information games, such as probabilistic events or information sets. However, it is unclear how this model makes it easier to solve these games. In the following section, I will review many of the current techniques used to solve imperfect information games, and offer a path forward on what techniques are most promising for providing more general solutions to imperfect information games.

2.3 How to Solve for Equilibria

In perfect information games, it is possible to calculate Nash equilibria by searching the the game tree, and many types of search algorithms have been proposed for different categories of perfect information games. For instance, the minimax search algorithm forms a heuristic value approximation for each action at a given game state by traversing the game tree while the active player works to minimize their loss in response to all other players trying to maximize it.

Why does this technique not work in imperfect information games? In imperfect information games, at any given game state, the active player only knows what information set they are in. They do not know the particular history. Because the different histories in the same information set may be in very different branches of the game tree (as is visible in Figure 2.2), they likely lead to starkly different rewards. It is also insufficient to simply average expected value across all possible histories for the current information state, as the probabilities of being at different histories are not necessarily equal, and these probabilities can be reasoned about with better approaches.

The first algorithms used to solved imperfect information games were nearly all variations of an approach known as linear programming. Linear programming attempts to maximize a variable or vector (representing a strategy), subject to a number of linear inequalities. For

a two-player zero-sum game, standard linear programming computes the Nash Equilibrium of a game in time exponential in the number of information states in the game tree. The approach works by iterating over all possible deterministic strategies, which grow exponentially in the number of information sets of the game. Despite the expensive time complexity, linear programming is useful on small games, and later optimizations even reduced the time complexity to polynomial in the number of information states. However, linear programming is still not scalable, and in response, a wide variety of different algorithms have been proposed.

Reinforcement Learning (RL) is an approach that has historically scaled to very large game tree sizes for a variety of perfect information games, providing hope for a general solution. Unfortunately, RL methods, including deep RL, generally fail to converge to equilibria when applied to imperfect information games. There have, however, been exceptions to this observation. Neural fictitious self play (NFSP) has been able to converge to equilibria in large-scale games by aggregating samples of the game tree and representing strategies as neural nets in a model-free manner (Heinrich and Silver 2016). Another area of active research lies in first-order methods, which take a gradient-based optimization approach. Because these methods rely only on calculating the gradient of a function representing the rewards of strategies, they have some of the best theoretical time complexity bounds of all modern methods (Kroer et al. 2015). Unfortunately, they have had limited practical success.

Among the variety of possible algorithms that can solve large imperfect information games, the approach that I believe has the best chances of leading to true generality, and the approach I will focus on for the remainder of my thesis, is the regret-based approach. In the following chapter, I will outline the fundamentals of regret-minimization and explain why the technique is superior to the above methods for producing

generality in solutions. I will also introduce an algorithmic paradigm known as *counterfactual regret minimization* that applies the principles of regret minimization to extensive-form games.

3 Counterfactual Regret Minimization

When a person must make a decision in a situation of uncertainty, then later information reveals that the best course of action was different from the one chosen, it is normal for that person to feel an amount of *regret*. In this chapter I will provide an overview to regret minimization, then specifically explore how counterfactual regret minimization applies these ideas to extensive form games. After this algorithmic paradigm is made clear, I will discuss how to evaluate the quality of the strategies produced by algorithms through a metric called *exploitability*, then finally look at some of the current limitations preventing counterfactual regret minimization algorithms from serving as general solutions.

3.1 Regret

The game-theoretic concept of regret, following from the human experience, is defined to be the difference in utility between the action that was taken and the optimal action. While it is clear that minimizing regret should tend to lead to better strategies, the optimal method for minimizing regret is not immediately apparent. Additionally, there are different quantities that can be minimized that may lead to different strategies.

For example, investing money in the stock market does not minimize worst-case regret, but may minimize average-case regret.

In 2000, Hart and Mas-Colell proposed a regret minimization approach known as *regret matching* (Hart and Mas-Colell 2000). In abstract terms, a player using regret matching will choose an action. After having the reward for all actions revealed, the player will evaluate how much they regret the action they took in their action sequence by subtracting the reward of their chosen action from the rewards of all other actions. This value for each action, the regret, is positive if a player regrets not choosing that action.

According to regret matching, the next time a player encounters that game state in a subsequent playing of the game, they will select each available action with probability proportional to the **positive regret** of that action. That is, for all actions with positive regret, the probability of choosing that action will equal the regret of that action divided by the sum of all positive regrets. For example, consider a game state with three available actions, A, B, and C. If the respective regrets of these actions are calculated as -1, 1, and 2, then a player using regret matching will select action A with probability 0, action B with probability $\frac{1}{3}$, and action C with probability $\frac{2}{3}$. Using this approach, the most-regretted actions have a higher probability of being selected, but the strategy is also not deterministic, which prevents subsequent moves from being predicted and exploited by an opponent.

Regret minimization is so named because using strategies like regret matching causes a player to minimize their regret in future playthroughs of the game. Traditionally, regret minimization has been applied to games such as the bandit problem rather than extensive-form games. However, the creation of counterfactual regret minimization in 2007 led to a tractable method to apply ideas of regret minimization to extensive-form games (Zinkevich et al. 2007).

3.2 The CFR Algorithm

3.2.1 Overview

The fundamental idea of counterfactual regret minimization (CFR) is to represent overall regret as the sum of many additive terms. Then, by minimizing these additive terms independently, overall regret can be minimized as well. This section is concerned with explaining the details of how this algorithmic process functions.

Because CFR works on extensive-form games, the algorithm cannot have a single global regret value. There must be a way to distinguish different game states. Rather than storing information about regrets for each possible action sequence, however, CFR instead stores regrets for each information set and subsequent action. So, CFR will have a tabular representation of regrets. For each information set in the game, and for each valid action at that information set, CFR will store and continually update the *cumulative regret* of choosing that action at that information set. The cumulative regret is the sum of the regrets for that information set/action pair at all iterations. Because the strategy is created by performing regret-matching on these tabular regrets within each information set, an upper bound on the total regret of the strategy can be generated by summing the regret at each information set/action pair.

These terms (the regrets at an information set/action pair) are referred to as counterfactual regrets. The *counterfactual regret* for choosing action a at information set I intuitively represents how much a player regrets using their current strategy, called σ, rather than a strategy that is identical to σ at all information sets except for I, at which it selects action a with probability 1. This counterfactual regret is represented as $r(I, a)$.

An agent will gradually learn the optimal strategy by performing a large number of *iterations*, where each iteration corresponds to travers-

ing the game tree a single time. This is often referred to as "self-play", as the algorithm plays against itself (though exploring every history rather than selecting one as in normal gameplay). In each iteration, there is a forward pass, during which each action at each history is explored until every terminal history has been reached. Then, after the traversal is complete and the utility payoffs at each terminal history are revealed, there is a backward pass, in which the revealed utilities are used to calculate the counterfactual regret at each information set/action pair. The appropriate tabular counterfactual regret value is then updated based on this result so the table's value reflects the *cumulative* counterfactual regret at that information set/action pair across all iterations. There are different flavors of CFR that handle regret updating among the players differently, though all have similar results. For this thesis, I use an *alternating* method, in which at each iteration, the game tree is traversed once per player. The respective player is then defined as the *regret-updating player*, and that player updates the global regrets only at game states where they are the active player. In the following iteration, all players will use the updated strategy generated from these updated regrets. The approach for this is outlined in Subsection 3.2.3.

After this training phase, the agent is able to play the game based on their strategy, which is defined by the tabular counterfactual regret values. The player's strategy follows the previously discussed *regret-matching* technique, in which the strategy is defined proportionally to positive regrets. If all regrets are non-positive, an action is sampled at uniform random. By repeatedly traversing the game tree during training, the *average* strategy (averaged over all iterations) is proven to converge to the Nash equilibrium, and this approximation of the Nash equilibrium tends to improve with more iterations.

3.2.2 Averaging

This is a lot of new information, and I will soon provide a mathematical grounding for this algorithm to make it more concrete. Before that, however, a simple sanity check is warranted: why are we concerned with the *average* strategy rather than the most recent? Why use cumulative regrets rather than the most recent regrets?

It is true that the most recent iterations tend to be better models of the game that the first several iterations. However, beyond the fact that it is the average strategy that is proven to converge to the Nash equilibrium, there are several intuitive reasons why averaging is important.

Firstly, averaging strategies and using cumulative regrets allows us to represent mixed strategies. We can demonstrate this by returning to rock-paper-scissors (though the analogy is imperfect because the game is not sequential). If the strategy were set according to the most recent regrets at each iteration, no matter how many iterations were conducted, after a loss the strategy would always demand choosing a single action (the action that would have won the previous round) with probability 1, because this is the only action with a positive regret. This is clearly not the game's equilibrium. Instead, averaging the strategy over many iterations using cumulative regrets would lead to the equilibrium of an equal probability of choosing each action.

Secondly, averaging prevents your opponent from exploiting any single deterministic policy you might have. For example, suppose you are playing a game of poker, and discover that you have been given the best hand in the game. While you may be tempted to bet a large amount of money, an opponent could exploit that strategy by folding (giving up without betting anything) whenever you bet large amounts. Averaging strategies ensures that the probabilities of selecting any action at a given information set are perfectly balanced so an opponent can never exploit the chosen action. This need to prevent exploitability also justifies why

the strategy dictates selecting actions proportionally to positive regret, rather than simply deterministically selecting the action with the largest counterfactual regret.

3.2.3 Updating Regrets

At this point, we are well-equipped to better understand how CFR calculates regrets to generate an optimal strategy. While we understand the general approach of the algorithm, in this section we will define exactly how regrets are updated during the reverse traversal of the game tree.

Recalling our representation of extensive-form games, $\boldsymbol{u}$ is a utility function that maps each terminal history of the game tree to a vector representing the reward that each player receives. Let $\boldsymbol{u[i]}$ denote the payoff to the *regret-updating* player i. We can then define the *counterfactual value* at a non-terminal history to be equal to the expected payoff from following our strategy σ. This is done by summing the the player payoff $\boldsymbol{u}[i]$ for each terminal history multiplied by the probability of reaching that terminal history (noting that some terminal histories may be unreachable). Finally, let us define $\pi^{\sigma}(h, h')$ to be the probability of reaching history h' from history h following strategy σ, and let $\pi^{\sigma}_{i-}(h, h')$ be the probability of reaching history h' from history h following strategy σ when only considering the probabilities of players other than i (i.e. player i can be thought of not acting according to σ, but to reach h'). This models the probability that an opponent will allow you to reach a particular game state. Recall $\boldsymbol{Z}$ is the set of terminal histories and let h_0 be the root history of the game tree. Then, we can formally define the *counterfactual value* at history h with a given strategy σ as:

$$\boldsymbol{v}[i](\sigma, h) = \sum_{z \in \boldsymbol{Z}} \boldsymbol{u}[i](z) \cdot \pi^{\sigma}(h, z) \cdot \pi^{\sigma}_{i-}(h_0, h)$$

To break this down: the counterfactual value of a node for a given player is dependent on the strategy employed, and this value can be found by summing over all terminal histories the following quantity: the payoff to the regret-updating player at that terminal history, multiplied by the probability of reaching that terminal history from the current history when playing according to strategy σ, multiplied by the probability of chance and your opponent allowing you to reach the current history from the beginning of the game. The first two terms in this product define the expected value given you are at history h, while the third term scales this expectation by the likelihood of your opponent letting you reach this history. This third term in the product is important, as a node that has an extremely high expected value will often be inaccessible since your opponent, using strategy σ as well, will generally not allow you to visit it.

With a definition for counterfactual value, it is simple to create a rigorous definition for counterfactual regret at a given history. Let $\sigma[I \mapsto a]$ refer the the strategy identical to σ, except at information set I, action a is chosen with probability 1.

$$r(h,a) = \boldsymbol{v}[i](\sigma[I \mapsto a], h) - \boldsymbol{v}[i](\sigma, h)$$

This matches our earlier non-formal definition of counterfactual regret: the difference in expected utility between the strategy that always chooses a at I, and the true strategy. However, a counterfactual value at a given *history* is not very useful, since a player does not know which history they are in when they are playing the game. To remedy this, we can define counterfactual regret at an information set:

$$r(I,a) = \sum_{h \in I} r(h,a)$$

which matches our definition of counterfactual regret proposed in earlier Subsection 3.2.1. Note that by summing the regrets for all histories, we are providing a maximum bound on total regret in the strategy.

We have learned that CFR uses a tabular representation to store the cumulative counterfactual regrets for each information set/action pair. Letting $r^t(I,a)$ represent $r(I,a)$ at iteration t using strategy σ^t, the cumulative regret at iteration T, $R^T(I,a)$ (which will be stored in the table) is then:

$$R^T(I,a) = \sum_{t=1}^{T} r^t(I,a)$$

At this point, the iteration has concluded, the table has been updated, and we can now use our updated cumulative regrets to update our strategy from σ^T to σ^{T+1}. We do this following the strategy of *regret matching* as described in Section 3.1, where the probability of choosing each action is proportional to that action's positive regret. The probability of choosing action a at information set I, is given by:

- if $R(I,a)$ is positive, the probability is equal to $R(I,a)$ divided by the sum of all positive cumulative regrets for all actions at information set I.
- if $R(I,a)$ is non-positive, but $\exists a' \in \mathcal{A}(I)$ such that $R(I,a')$ is positive, then the probability is 0.
- if $R(I,a)$ is non-positive and $\nexists a' \in \mathcal{A}(I)$ such that $R(I,a')$ is positive, then the action is chosen uniform randomly.

We can represent this formally as:

$$\sigma^{T+1}(I,a) = \begin{cases} \frac{\max(0,R^T(I,a))}{\sum_{a \in \mathcal{A}(I)} \max(0,R^T(I,a))} & \text{if denominator} > 0 \\ \frac{1}{|\mathcal{A}(I)|} & \text{otherwise} \end{cases}$$

After this step, we have finally updated our strategy (keep in mind we are storing our converging average strategy as well), and we are ready to start iteration $T + 1$. When we have concluded the desired number of iterations, we can return our average strategy, and CFR is complete.

3.3 How To Measure Success?

Once CFR is complete and we have a returned strategy, how should we measure how good a strategy is?

There are many possible answers. We could compare two strategies by having bots each adopt one of the strategies and play many games against each other, checking if one tends to win more than the other. Similarly, we could force these bots to both play against the same opponent that has a standardized strategy.

These approaches are useful, and are actively used to compare different algorithms for solving imperfect information games today. However, approaches like these do have some disadvantages. It can be difficult to quantify an absolute metric for quality of the strategy, and furthermore, since different strategies may vary in how strongly they exploit the mistakes of their opponent, the win/loss ratio may not even serve as an accurate correlate of strategy quality.

Rather than relying on gameplay, ideally, we would want a method to measure a given strategy for closeness to the Nash equilibrium of the game. Unfortunately, for many games, the Nash equilibrium is unknown. However, we can approximate this closeness through a metric known as *exploitability*, which is the key metric used to evaluate strategy optimality.

The exploitability of strategy σ measures by how much more the *Best-Response* strategy is able to beat σ relative to the Nash equilibrium strategy, averaged across all player positions. It is calculated as the ex-

pected loss per game to player i given player i plays by strategy σ, and the opposite player plays by strategy $B(\sigma)$. The Nash equilibrium is defined to have an exploitability of 0, and so the smaller our strategy's exploitability, the closer we are to the Nash equilibrium.

The Best-Response strategy, $B(\sigma)$, is defined as the strategy that achieves the highest expected value when played against σ. That is, the Best-Response strategy has access to σ, and knows the probabilities with which its opponent will choose any action at any information state. Thus, the Best-Response strategy can deterministically choose their actions to minimize the expected value of their opponent. The Best-Response strategy is a pure strategy in which the action that minimizes the expected value of the player using σ should be chosen with probability 1, as measured by the *counterfactual values* of the different information set/action pairs available.

To put this more formally, let σ_i be a strategy used by player i, let σ_i^N be a Nash equilibrium strategy used by player i, and let $U_i(\sigma_i, \sigma_j')$ be the expected utility payoff to player i playing with strategy σ_i against any number of opponents playing with strategy σ_j'. Then, the exploitability for player i is given by:

$$\epsilon_i(\sigma_i) = U_i\Big(\sigma_i^N, B\Big(\sigma_i^N\Big)\Big) - U_i(\sigma_i, B(\sigma_i))$$

and total exploitability for the strategy σ averaged across all player positions with player set of $\mathcal{N}$ is given by:

$$\epsilon(\sigma) = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \epsilon_i(\sigma)$$

3.4 Pros and Cons

While CFR is an extraordinary algorithm, there are some limitations to using CFR as a general method for solving imperfect information games,

and there are two principal problems for generality in the standard CFR algorithm presented.

In CFR, the entire game tree must be traversed on each iteration. This task is impractical for games with extremely large game trees (e.g. Stratego). Other techniques, though inferior in practice (such as reinforcement learning), need only to sample a single path in the tree. A general solution should not be restricted by the shape or size of the game tree.

Secondly, and partially as a consequence of the first limitation, CFR cannot be applied to large games due to unreasonable requirements for both time and memory. As a result, making a large game tree game solvable via CFR requires abstraction. Abstraction algorithms take a game tree as input, and produce a substantially smaller game tree out as output to serve as an "abstraction" of the game with many game states joined together into single states. While this allows for CFR to be applied to a much broader set of games, it hinders generality for a number of reasons. Firstly, this is a less faithful game representation, and courser abstractions allow for faster convergence but the resulting strategies are more exploitable and often don't translate perfectly to the original game. Perhaps even more importantly, good abstraction algorithms require prior knowledge of the game's equilibrium. While this is not a problem in extremely well-studied games like poker, this task makes it infeasible to apply CFR to novel large games.

Base on these limitations, a fundamental hurdle that CFR faces in becoming more general is its current inability to perform well on large or nonstandardly structured game trees.

3.4.1 So Why CFR?

The central aim of this thesis is to highlight a path forward in producing generality in methods of solving imperfect information games. I have

narrowed my focus to regret-based approaches and CFR in particular, and the choice to promote this single category of algorithms as the path toward generality despite its limitations requires justification.

Regret-based approaches, and in particular CFR, have a variety of advantages relative to the other methods of solving imperfect information games mentioned in Chapter 2.

First and foremost, minimizing regret definitionally minimizes exploitability (Zinkevich et al. 2007). This is extremely important for achieving generality. In the early days of solving imperfect information games, linear programming techniques were widely used and often maximized expected value. At first, algorithms trained by these techniques on poker consistently beat professional human players; however, human players soon learned how to exploit the poker bots' strategy. By minimizing regret, it is guaranteed that no opponent could take advantage of your strategy.

Additionally, CFR is the algorithm that has generated nearly all of the top-performing strategies in imperfect-information games (Brown and Sandholm 2019b)(Moravčík et al. 2017). Additionally, among algorithms that converge to the Nash equilibrium, while CFR does not have the best theoretical convergence, in practice it has the best empirical convergence by a substantial margin. It is true, however, that the majority of algorithms for imperfect-information games have been tested on largely on poker, and this narrow scope may limit the conclusions we can draw about the generality of regret minimization.

Finally, and perhaps most importantly, CFR as an algorithmic paradigm is remarkably plastic, and has given rise to an entire "CFR family" of algorithms that have been fine-tuned optimize for a variety of different properties. In the following chapter, we will see how this plastic nature of CFR has led to the generation of CFR variants that overcome the highlighted limitations above.

4 The New Kids on the Block: CFR Variants

While CFR has revolutionized performance in calculating equilibria in imperfect information games, it nonetheless has a variety of weaknesses that prevent it from being a truly general solution. However, as claimed in the previous chapter, the extraordinary flexibility of the CFR paradigm allows for the creation of CFR *variants* — modifications to the original algorithm (which is commonly referred to as *vanilla CFR*) that modify its mathematical properties or performance.

In this chapter, I will highlight some modifications to the CFR algorithm that have been recently proposed that resolve the main limitations of the vanilla CFR algorithm mentioned in Section 3.4.

4.1 Discounted CFR

This variant, in contrast to the other two, does not address a philosophical limitation with vanilla CFR. However, its role in increasing empirical performance has been so central in recent literature that it is still worth mentioning briefly.

In vanilla CFR, each iteration contributes equally to the cumulative regret stored in a table. However, it is likely the case that some iterations (most notably the first iterations), contribute far less accurate approximations of regret. As a result, the algorithm must run for more total

iterations to allow the cumulative regrets to converge to more accurate ratios. This inefficiency led in 2019 to the development of a family of algorithms known as **Discounted CFR** (Brown and Sandholm 2019a).

Discounted CFR works by assigning a weight to each iteration, and the influence of that iteration on the updates to the cumulative regrets is scaled by that weight. We discussed in the previous chapter why using only the most recent regrets is not a feasible approach (it leads toward exploitability and doesn't support mixes strategies as well), but by using this weighting approach, one could employ discounted CFR to more heavily weight later iterations of the algorithm, as later iterations tend to have more accurate regret approximations. Indeed, one of the most popular and successful variants of discounted CFR is known as *Linear CFR*. In linear CFR, the weight given to each iteration increases linearly throughout training. That is, the updates to cumulative regret at iteration t are weighted by the multiplicative factor t.

While simple in concept, discounted CFR has had substantial influence. In particular, by weighting contributions to positive and negative regrets differently via hyperparameters, discounted CFR has led to new state-of-the-art performance in a selection of games (Brown and Sandholm 2019a). While discounted CFR works by adjusting the weight of each iteration on updating cumulative regret, the average strategy is still computed using the method described in Subsection 3.2.3.

Discounted CFR has been instrumental in demonstrating the practical value of CFR techniques and the flexibility of the algorithm. Additionally, it inspires many of the ideas presented in Chapter 5 that are used to build more general algorithms.

4.2 Monte Carlo CFR

As discussed in the previous chapter, one of the most challenging problems in creating more general CFR techniques for imperfect information games is finding methods to converge to equilibria for games with trees that can be unboundedly large.

For example, consider the imperfect information game of no-limit poker. When played with 500-blind stacks (this refers to the amount of money available to bet), no-limit poker contains 7.16×10^{75} game states (Johanson 2013). Solving such a game with a vanilla CFR implementation would require 1.241×10^{49} yottabytes of memory, which is beyond unreasonable.

For contrast, approximately 4.48×10^{44} states exist in chess (Tromp 2021). This is an enormous difference. To put the magnitude of this difference in context, it is estimated that there are approximately 10^{55} atoms in the earth, while there are an estimated 10^{80} atoms in the entire universe.

Because it is unreasonable to traverse the game tree for many different varieties of games, Monte Carlo CFR (MCCFR) was developed to introduce sampling methods that dramatically reduce the size of the traversed area of the game tree (Lanctot, Waugh, et al. 2009), making CFR methods much more generally applicable to different tree shapes and sizes.

Monte Carlo methods refer to a broad class of statistical methods for modelling uncertainty via random variables. In MCCFR, we use this technique at certain information sets to sample a particular action according to a probability distribution rather than traversing every single action. Although this sampling approach generally requires more iterations to reach the same exploitability as vanilla CFR, the astonishingly lower cost per iteration often leads to much faster convergence to the Nash equilibrium.

There are many methods for performing this sampling, and here I will focus on two variants of MCCFR that are especially relevant for my contributions in the following chapter: Outcome Sampling and External Sampling.

Outcome sampling is a sampling method that requires only a single action to be sampled at every information set encountered in an iteration, ultimately leading to a single terminal history. Following this, the tabular cumulative regrets for each of the sampled information set/action pairs will be updated.

This sampling is performed by choosing each action with probability equal to the probability that action is chosen under the current strategy. So, if we let $\sigma^t(I, a)$ represent the probability of choosing action a at information set I at iteration t according to current strategy σ^t, in outcome sampling we will sample $a \in \mathcal{A}(I)$ with probability $\sigma^t(I, a)$.

Outcome sampling has a variety of extraordinary properties. A normal traversal in vanilla CFR has computational complexity linear in the tree size, while outcome sampling has complexity logarithmic in tree size (in other words, vanilla CFR is exponential in tree depth and outcome sampling is linear in tree depth). Considering a tree with branching factor 10 and depth 10, a single iteration in vanilla CFR requires visiting over one billion game states ($\sum_{i=0}^{9} 10^i$), while a single iteration in outcome-sampled MCCFR requires visiting only ten game states. While outcome sampling requires more iterations to reach the same exploitability, it is generally able to do so far more quickly that vanilla CFR.

Additionally, because only a single action sequence is considered at each iteration, some recent work has explored applying reinforcement learning (RL) techniques to outcome-sampled MCCFR (Steinberger et al. 2020), which has the potential to introduce the expansive suite of RL tools into CFR algorithms.

External sampling is a blend of outcome sampling and vanilla CFR that is perhaps the most commonly used variant of MCCFR today. In External sampling, at all game states where the regret-updating player is the active player, all actions are traversed, just like in vanilla CFR. However, at all other nodes (the opponent or chance nodes), an action is sampled according to outcome sampling.

Consider our fictional free with branching factor 10 and depth 10. If there are two players who alternate turns, while vanilla CFR still visits over one billion nodes per iteration, external sampling MCCFR visits just over 22 thousand nodes with two players if the regret-updating player goes second ($\sum_{i=0}^{9} 10^{\lfloor \frac{i}{2} \rfloor}$, where $\lfloor x \rfloor$ is the floor function). This is still a dramatic reduction, and in fact, external sampling generally performs astoundingly better than outcome sampling, requiring both fewer iterations and less time to converge to the equilibrium. In fact, external sampling is used in almost all state-of-the-art CFR algorithms (Brown and Sandholm 2019b)(Moravčík et al. 2017). Despite this, external sampling does not have many of the same theoretical properties of outcome sampling.

While outcome sampling traverses a number of nodes that grows linearly with game tree depth, the number of nodes traversed in external sampling grows according to $\mathcal{O}\left(2^{\sqrt{n}}\right)$, where n is the tree depth. This is an improvement from the $\mathcal{O}(2^n)$ bound of vanilla CFR, but still possesses an exponential factor that limits its applicability to large trees. While some researchers have proposed the term "moderately exponential growth" to describe this rate of growth, I will continue to use the term "exponential" throughout this thesis.

Using the game of Kuhn poker as a model, Figure 4.1 illustrates a possible partial traversal when player 1 is the regret-updating player for both outcome sampling (yellow) and external sampling (blue) using their respective sampling strategies as defined above.

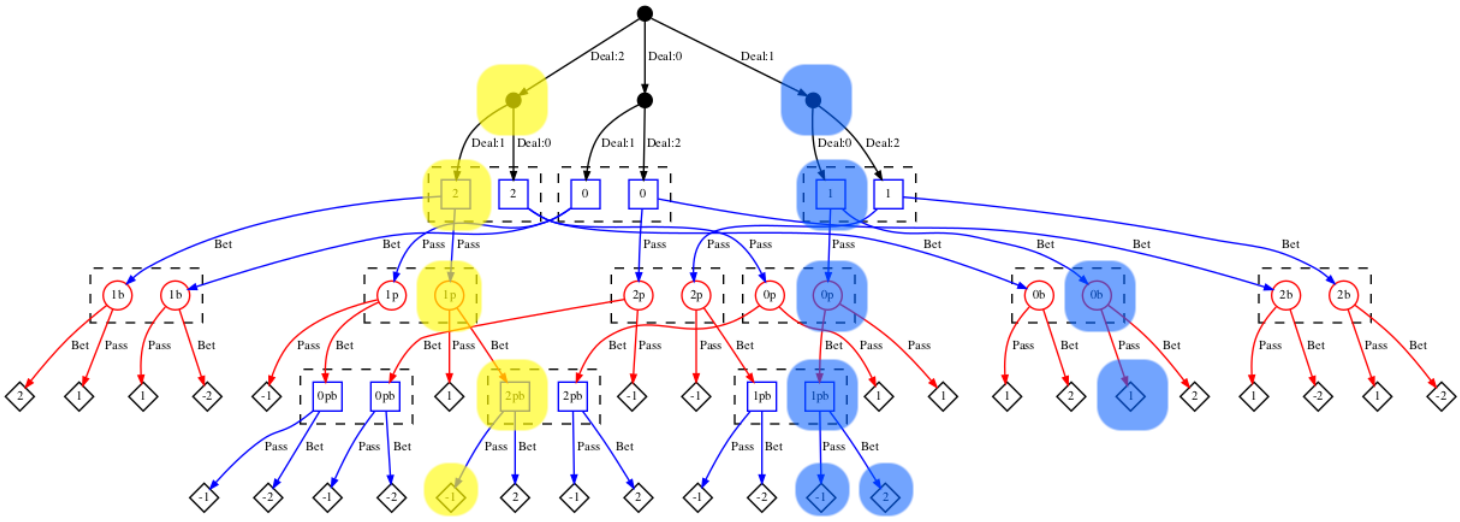


Figure 4.1: The game tree of Kuhn poker with information sets — highlighted to show a possible partial traversal for outcome sampling (yellow) and external sampling (blue).

An important aspect of MCCFR to be mindful of, despite its benefits, is that all varieties of MCCFR, including both outcome and external sampling, introduce small errors into the updated strategy. Because all actions that are not sampled contribute no counterfactual value to the strategy updates, there is a large amount of noise introduced to the convergence process, especially when a very valuable action is not sampled. As a result, as the algorithm converges to the Nash equilibrium near the end of training, the convergence can be stifled due to this noise, a phenomenon referred to as the introduction of *variance*. Fortunately, variance-reduction has been an extremely fruitful field of research in recent years, and methods to reduce variance multiple orders of magnitude have been introduced as a standard counterpart of MCCFR use (Schmid et al. 2019).

4.3 Deep CFR

In the previous chapter, I mentioned that using CFR to solve extremely large games often requires *abstraction*, putting the game tree through an abstraction algorithm to dramatically reduce its size. This can be harmful because it leads to both poorer and more exploitable strategies, but most importantly for developing truly general CFR, a good abstraction algorithm requires prior knowledge about the game's equilibrium, which is not possible for many novel games.

The final variant of CFR I will discuss, Deep CFR, is perhaps the most significant development toward generality in CFR in the past few years. Invented in 2019, Deep CFR obviates the need for abstraction by removing the table of regrets entirely, instead using a deep neural network to approximate the behavior of vanilla CFR on the full, unabstracted game (Brown, Lerer, et al. 2019). Neural networks have historically been extremely useful methods of implementing function approximation in large state spaces (particularly in deep reinforcement learning), and they are similarly useful when applied to CFR.

At each iteration of Deep CFR, the following steps are performed. First, a constant k partial traversals of the game tree are conducted, generally with external sampling. Following this, a neural network referred to as the "value net" called $\mathbf{V}$ is *trained from scratch*. At a high level, the purpose of this value net is to approximate the cumulative regret of taking each available action given an information set. The network takes a single information set as input, and outputs a vector of length $|\boldsymbol{\mathcal{A}}(I)|$, where each value in this vector corresponds to a single action in $\boldsymbol{\mathcal{A}}(I)$. These values for an information set/action pair are known as *advantages*, and they corresponds to the cumulative regrets for each action ($R^T(I,a)$) in vanilla CFR. To train the value net, a variety of approaches have been used, though the original approach minimizes mean squared error between predicted advantage and stored samples of instantaneous regrets

in all other iterations. These samples of instantaneous regret are stored in memory buffers unique to each player that we can call $\mathcal{M}_{\mathbf{V},i}$, where i corresponds to the player.

After an iteration is complete, Deep CFR then needs to produce a representation of the average strategy. To do this, a memory bank $\mathcal{M}_{\mathbf{\Pi}}$ is created, which stores information set probability vectors of the form $\sigma(I)$, or a vector containing the probability of choosing each action at $\mathcal{A}(I)$. During training, whenever an information set I belonging to player p is traversed during the opposing player's traversal of the game tree, $\sigma^t(I)$ is added to $\mathcal{M}_{\mathbf{\Pi}}$. If linear CFR is applied, it will be stored with weight t. At the conclusion of a run, a second neural network $\mathbf{\Pi}$ called the *policy net* will be trained on $\mathcal{M}_{\mathbf{\Pi}}$, after which $\mathbf{\Pi}$ represents the strategy to be used. $\mathbf{\Pi}$ can be used to generate the average strategy at a given information set; it takes an information set as its input, and returns a vector containing the probabilities of selecting each action at that information set according to the strategy.

To successfully use Deep CFR, multiple banks of memory must be managed: there will be n $\mathcal{M}_{\mathbf{V},i}$ banks, one for each of n players, which store sampled instantaneous regrets, along with the final memory bank $\mathcal{M}_{\mathbf{\Pi}}$, which stores sampled $\sigma(I)$ objects. Generally, these memory banks will be bounded in size. If a memory bank is filled during training, items can be sampled randomly for removal using a sampling approach such as *reservoir sampling*.

Deep CFR, while only given a cursory overview here, nonetheless greatly expands the philosophical properties of CFR solutions. Two extremely strong advantages in particular are: 1. Deep CFR removes the need for abstracting games, allowing CFR for the first time to solve large novel games, and 2. One no longer needs to visit information set I to update $\sigma(I)$, as the neural net is able to construct these associations.

4.4 Where Does CFR Stand Today?

The three variants of CFR highlighted here all contribute greatly to resolving the factors limiting generality in CFR highlighted in Section 3.4, and taken together, my hope is that these CFR variants illustrate the potential of CFR in leading toward general solutions for imperfect information games. Unfortunately, even these modern CFR variants today possess some limitations preventing them from fully achieving this goal.

Importantly, while MCCFR allows for the partial traversal of larger game trees, the current best-performing sampling strategy, external sampling, is heavily optimized for the subset of games with small branching factors. In particular, the centrality of poker as a model game in the imperfect information space has led to an over-optimization for the game tree of poker. In addition, extant sampling methods are often inefficient in which information sets are sampled. In the following chapter, I will resolve this dilemma by introducing a new sampling scheme that can be applied to game trees of any shape, including game trees that external sampling is not practical for due to high branching factor.

Additionally, while Deep CFR is an excellent move forward in generality of CFR techniques, the high cost of retraining the value net at every iteration makes it impractical for many settings. As a result, Deep CFR is not currently used in any top-performing CFR models. In Chapter 6, I will introduce a new framework that allows for the potential removal of part of this cost, demonstrating that Deep CFR could provide a general solution to problems that historically other methods have been used to solve.

While these discussed CFR variants certainly mark large steps in improving the generality of CFR techniques, these two limitations mark what I see as the two largest obstacles to CFR being a general solution to imperfect information games. In the following two chapters, I will highlight these two mentioned areas where these novel algorithms fall

short in their generality, proposing my own CFR variants that directly address these limitations.

5 Two New General Sampling Methods

5.1 The Philosophy behind Sampling

As discussed in the previous chapter, one of the most difficult problems in effective CFR algorithms is finding methods to traverse game trees that can be extremely large or unconventionally shaped.

Because of the centrality of poker as a model game for imperfect information systems, many CFR variants are optimized for the shape of the game tree of poker-style games. However, poker has a game tree with small branching factor that is not representative of the structure of many other imperfect information games. Since our ambition is to develop CFR methods that can contribute to problem solving in real-world imperfect information settings, optimizing algorithms to the shape of one particular game hinders generality.

There are a variety of different ways that trees can grow. Two of the most important when considering CFR algorithms are the game's *branching factor* and *depth*. A tree with large branching factor has a large number of different actions available to the active player at each game state, while a tree with a large depth corresponds to a game with a large number of actions taken per game.

External sampling, the current best-performance sampling technique (far superior to outcome sampling), actually performs extremely ineffi-

ciently or fails entirely on games with large branching factors, leaving outcome sampling as the only currently available option. This leaves an enormous gap in current knowledge surrounding how to traverse such large or nonstandardly shaped game trees, and little work has been done to fill this gap. There are a few exceptions to this; for example, some recent work in 2018 has proposed a method of *robust sampling*, which is identical to external sampling, but rather than traversing all action paths at the node of the set player, a pre-fixed number k is defined, and at each node of the set player, k paths are traversed (Li et al. 2018). Approaches like this are a step in the right direction, but are still limited, as they do not avoid exponential growth and are not taking advantage of knowledge of the distribution of information.

Sampling, at its core, is a tool that tries to understand the game broadly while only interacting directly with a smaller subset of the game. It should be used intelligently to sample only the information that is needed to find the Nash equilibrium. Current sampling strategies rarely employ this idea, instead defining *patterns* of sampling that approximate the game tree as a whole.

The value of information in different loci of the game tree at different times may be described informally by the *Pareto Principle*: sampling a very small number of areas may be responsible for a very large quantity of relevant information for determining the optimal strategy. In other words, the importance of information at different locations in the game tree and at different points in time may be distributed according to a power-law distribution.

Sampling should attempt to identify the highest-importance information by responding to a variety of factors rather than using a uniform sampling approach. In particular, I propose achieving this by:

1. Varying sampling technique by iteration.

2. Varying sampling technique by locus in the game tree.

I have not seen any evidence of either of these techniques ever being utilized in CFR, and I believe that sampling according to the importance of information to be gained may be the most promising path to creating CFR algorithms that generalize to varied shapes of game trees and different classes of imperfect information games. To address this, I developed two classes of sampling techniques that work by approximating the value of information at different states or times. The first varies sampling strategy with iteration, while the second varies sampling strategy with locus in the game tree. The ultimate goal of such methods is to introduce a novel sampling strategy with the performance of external sampling that can be generalized to large games that are currently only traversable by outcome sampling.

After writing and implementing variations of these two algorithms, I computed their exploitability in comparison to pure OS and ES strategies. I used Deepmind's Openspiel implementations of OS and ES to compare exploitability among the different approaches (Lanctot, Lockhart, et al. 2019).

5.2 Varying by iteration

At different iterations in the training process, different information is known about the game tree. So, a more optimal sampling algorithm may be found by varying sampling technique between iterations, following the basic principle driving discounted CFR.

In particular, we could create a *mixed* sampling strategy in which the high-performance external sampling (referred to hereafter as ES) is used to quickly generate a coarse approximation of the Nash equilibrium, which can then be traversed efficiently with outcome sampling (OS), which will with high probability visit the nodes identified by ES to be more likely to be part of an optimal strategy. In this sense, we can

manage an *exploration-exploitation* trade-off by using ES to explore the game tree, and the much more efficient OS to exploit action sequences likely to contain important information.

As a quick note: this concept of *exploitation* can easily be confused with the previously established concept of *exploitability*. While I will continue to use both terms, it is worth emphasizing that exploitation refers to a sampling strategy taking advantage of information, while exploitability is a metric that measures the closeness of a strategy to the Nash equilibrium.

As an elementary example, following the linear intuition behind the successful linear CFR, consider a sampling method that linearly decreases the probably of using ES in a given iteration from 1 to 0 over the course of training. That is, given a uniform random number x in range $[0, 1]$ each iteration, we will employ ES if $x > \frac{\text{current iteration}}{\text{total iterations}}$. In this way, near the beginning of training, when little information about the strategy is known, we will be more likely to explore via ES, and near the end of training, when the optimal strategy is beginning to take form, we will exploit sample paths nearer to this optimal strategy via OS.

To measure the performance of this sampling method, we can compute the exploitability (see Section 3.3) of the average strategy after each iteration, leading to a demonstration of how quickly each sampling algorithm converges to zero exploitability. A graphical illustration of this is demonstrated for the game of Kuhn poker in Figure 5.1. In this toy example, external sampling is clearly superior to outcome sampling when measured according to iteration.

Because the small tree size of Kuhn poker limits the usefulness of comparisons and introduces a substantial amount of noise, I will be comparing benchmarks on two different games. *Leduc Hold'em* (a simple poker variant with reduced deck size and limits on betting amounts) will be used to model performance on poker-style game trees, while

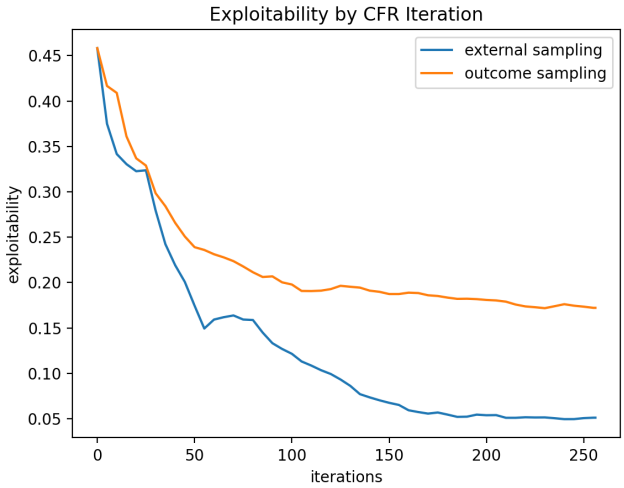


Figure 5.1: A single CFR training session of Kuhn poker consisting of 2^8 iterations for both external and outcome sampling. This toy example illustrates how the exploitability of the average strategy can be plotted over time.

Liar's dice (a game involving predictions about the results of dice rolls) will be used to model performance on larger game trees with large branching factors.

Additionally, I propose measuring exploitability as a function of nodes touched during traversal rather than as a function of iteration number. External sampling, as it traverses every node touched by outcome sampling in addition to many more during a single iteration, will necessarily show faster convergence on a per-iteration basis. However, the iterations of ES are exponentially more computationally expensive than those of OS in the number of visited game states. Because we aim to measure the efficiency with which low exploitability can be reached, exploitability as a function of nodes touched is a much more useful metric. Another option could be to measure exploitability as a function of time, but due to possible variance in implementation of the relevant algorithms I decided nodes touched would be more informative.

I tested the performance of variations of the proposed mixed sampling method that varies by iteration on both Leduc Hold'em to understand the implications of such a sampling method on poker-style games. In particular, I tested:

- performance of the mixed sampling method in which the probability of selecting external sampling decreases linearly over all iterations
- performance of the mixed sampling method with an *exponential* decrease in probability of selecting external sampling.
- performance of the mixed sampling method that re-selects a sampling method at every visited node, rather than once for the entire iteration.
- performance with the introduction of an explicit exploration vs. exploitation hyperparameter when outcome sampling is used.

5.2.1 Results on Poker-style games

After measuring the performance of a number of different sampling algorithms that vary by iteration, the results indicate that a linearly decreasing mixed sampling strategy can match the performance of pure external sampling on Leduc Hold'em (the current best sampling method). Additionally, the exponentially decreasing mixed strategy, while inferior in some settings, nonetheless performs far better than outcome sampling while having the same computational complexity. As such, this technique could offer dramatic performance improvements in large games that ES cannot be used for. For context, Leduc Hold'em has a average branching factor between 2 and 3, and the average depth across all terminal histories is 16.31. While these results will be discussed individu-

ally, an overview of the different variations of this sampling strategy can be seen in Figure 5.2.

Linear

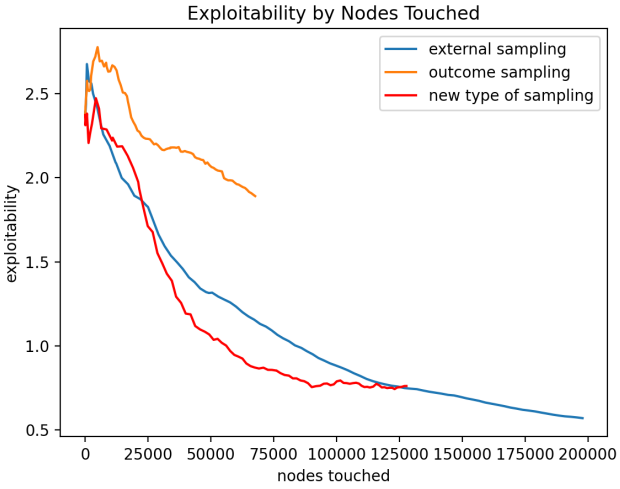
Before conducting these tests, I anticipated that a mixed sampling method with linear decrease in probability of selecting ES would perform better than simply averaging external and outcome sampling methods together due to OS being used later in the training process, exploiting the already partially-optimized strategy. Not only is this true, but Figure 5.2a indicates that this mixed sampling method matches performance of ES on Leduc Hold'em.

While I hoped that this sampling of strategy-relevant nodes by OS would increase the average convergence per node touched, this mixed sampling approach does not appear to offer any significant benefit over ES in Leduc Hold'em.

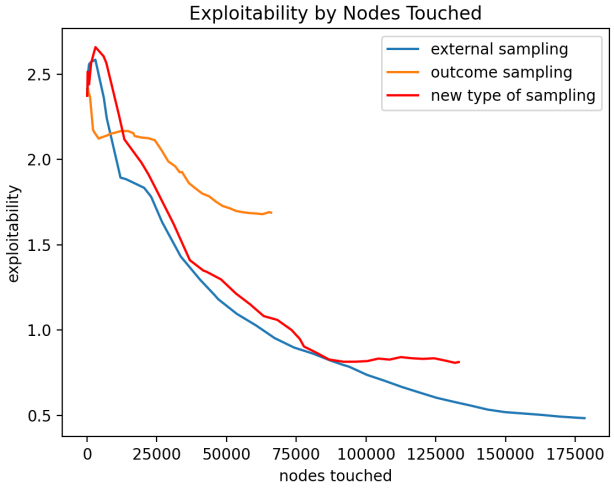
Exponential

While the linear decrease in probability of selecting ES matches the performance of pure ES, it doesn't solve the problems regarding the time complexity of ES. This linear approach reduces the time complexity of the sampling method by only a constant factor in the number of iterations (the expected number of iterations will be the arithmetic average of the expected number of iterations for pure OS and ES). Ideally, we would want to decrease this complexity exponentially in the number of iterations instead. We can do this by decreasing the probability of selecting ES **exponentially** rather than linearly in the number of iterations.

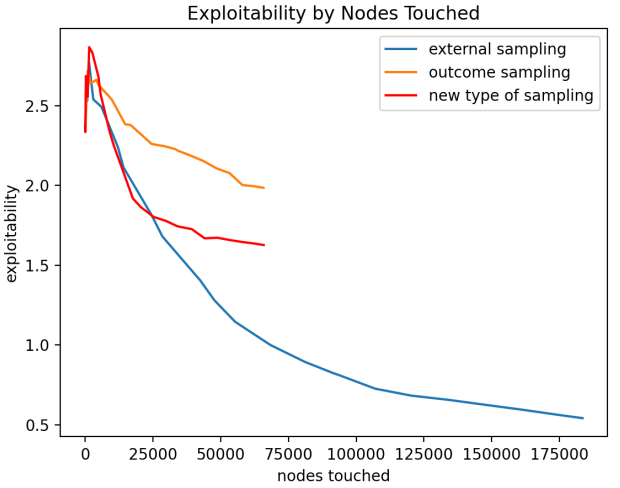
Specifically, we can define a halflife H. On each iteration i, we can generate a uniform random number x in range $[0,1]$, then employ ES if $x < 2^{(-\frac{i}{H})}$. Using this method, the probability of using external sampling will be cut in half every H iterations. Because H does not vary with the



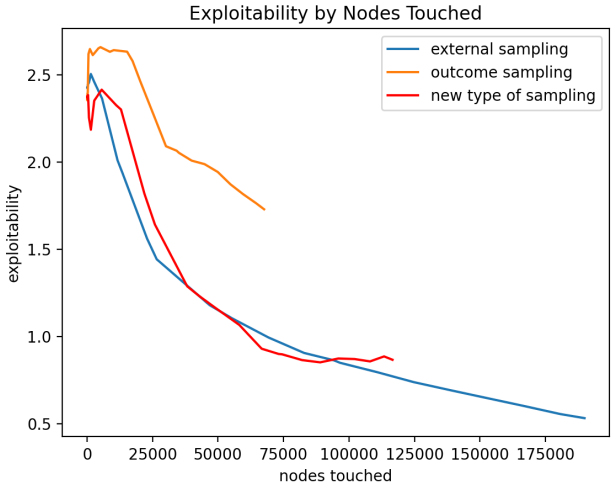
(a) Linear Decrease of p(ES) from 1 to 0



(b) Exponential decrease of p(ES) with halflife $2^{10} = 1024$



(c) Exponential decrease of p(ES) with halflife $2^3 = 8$



(d) Linear Decrease of p(ES) from 1 to 0, re-selecting sampling method at each node

Figure 5.2: Four variations of a mixed sampling strategy in which either ES or OS is selected to be used at each iteration. The probability of selecting ES, p(ES), varies with iteration based on a different functions. All trials took place over 2^{12} iterations on Leduc Hold'em.

number of iterations, the time complexity of such a sampling method can be linearly bounded in game-tree depth! The iterations at which ES is used will still have an exponential in depth cost, but because the ES iterations are occurring spaced logarithmically across time, the total complexity can be bounded linearly, like outcome sampling. In other words, the computational complexity both this exponential approach and OS would grow logarithmically with the size of the game tree, while that of ES would grow linearly.

We can additionally estimate the percentage of iterations for which ES is used, which we can call $Per(\text{ES})$. Letting i be the current iteration step, I the total number of iterations, and H the halflife measured in iteration count, we have:

$$Per(\text{ES}) = \frac{\sum_{i=1}^{I} (2)^{\left(-\frac{i}{H}\right)}}{I}$$

Unfortunately this exponentially decreasing mixed sampling strategy is unable to match the performance of ES. In Figure Figure 5.2b, $Per(\text{ES}) = .338$, and performance is comparable but slightly inferior to ES. However, to bound the complexity of this sampling method in linear time in tree depth, H cannot increase faster than logarithmically in the number of iterations performed. So, to model situations where $H \ll I$, Figure 5.2c illustrates an example in which $H = \frac{I}{512}$.

In this example, performance is notably inferior to pure ES, indicating that the exponential factor may not be able to be removed entirely while matching the performance of ES. However, in Figure 5.2c, $Per(\text{ES}) = .0027$, an extremely small fraction, with OS accounting for the sampling method used on nearly all iterations. Despite this, this mixed strategy still shows a dramatic improvement relative to pure OS. This indicates that this exponentially-decreasing sampling method (linearly bounded in the tree depth) could result in dramatic performance improvements compared to OS on games too large for ES use. So, this

exponentially decreasing sampling method could offer enormous performance increases in applying CFR to large games.

Selection by Node

For all previous experiments, we have selected a sampling method at the start of the iteration, and used this method to traverse the game tree. However, it is also possible to select a new sampling method at each individual node in the game tree. In Figure 5.2d, I followed the same linear decrease proportional to iteration method as in Figure 5.2a, but re-selected a sampling strategy at each node in the tree rather than once-per iteration. Performance appears to be relatively unchanged relative to the standard linear method, and indeed, ES and OS are still selected at the same ratio. However, this approach introduces new possibilities surrounding different loci that will be explored further in the following section, Section 5.3.

5.2.2 An Explanation through an Exploration Hyperparameter

Throughout this chapter so far, I have philosophically justified the performance of these mixed strategy methods by referring to the ability of outcome sampling to exploit a partially-optimized strategy discovered by external sampling.

In a final experiment, I attempt to substantiate this claim by introducing an explicit exploration-exploitation hyperparameter λ into the OS algorithm.

λ, defined to be a number in the range $[0, 1]$ that stays constant for the duration of training, defines the probability of engaging in "exploration" by sampling uniformly each time a node is encountered. Specifically, at any node, if outcome sampling is used, the chosen action will be

λ	Exploitability
0	1.48
0.25	1.58
0.5	1.65
0.75	1.67
1	1.73

(a) Linear Mixed Sampling Method

λ	Exploitability
0	2.22
0.25	2.22
0.5	2.20
0.75	2.34
1	2.37

(b) Pure Outcome Sampling

Table 5.1: Exploitability of strategies produced by two sampling algorithms for Leduc Hold'em, trained for 2^{10} iterations with varying exploration hyperparameters. λ indicates the probability that an action at a given node will be selected at uniform random rather than using the standard outcome sampling procedure.

sampled uniformly (exploration) with probability λ, and will be sampled according to the strategy (as in normal outcome sampling) with probability $1-\lambda$. The standard outcome sampling I introduced in Chapter 4 has $\lambda = 0$, while an approach that samples all actions uniformly randomly has $\lambda = 1$. Table 5.1 shows the exploitability after training of average strategies produced by both a per-node linearly-reduced mixed sampling method (the same method employed in Figure 5.2d), and pure outcome sampling.

These results support the justification that outcome sampling offers improvement to the tested mixed strategies due to its ability to exploit approximated strategies. In pure outcome sampling, increased exploitation (smaller λ) is unable to offer improved performance past a certain threshold ($\sim \lambda = .5$.) In contrast, in our mixed approach, higher levels of exploitation continue to offer increased performance until $\lambda = 0$. This supports the claim that outcome sampling leads to good performance in our mixed sampling method by exploiting a partially-optimized strategy.

5.2.3 Section Review

At this point, we have demonstrated a few critical insights about the performance of sampling strategies that vary in the probability of selecting ES or OS based on iteration number on poker-style games:

- A linear decrease in ES/OS ratio can maintain similar performance while reducing complexity by a constant factor.
- An exponential decrease in ES/OS ratio can reduce complexity by an exponential factor, allowing for applicability to large games. While inferior to pure ES on small games, this method produces substantially better results than the current best option OS on large games.
- The benefit found in this method is attributable to the ability of OS to exploit a partially-optimized strategy.

We can then propose a general mixed sampling strategy $\mathcal{M}(f)$. This strategy takes in a function f mapping from iteration number to the probability of using ES at that iteration, and uses this function to sample efficiently. Generally, f will be a decreasing function, and in this section we particularly discussed linearly decreasing (within defined bounds) and exponentially decreasing functions f can adopt.

5.3 Varying by Locus in Tree

In the previous section, we discussed how the optimal sampling method may vary with iteration. In this section, I will propose a new idea: that the optimal sampling method may vary with locus in the game tree. Some game states are uninteresting to traverse during training because they offer little information about the optimal strategy. For example, at a game state that clearly has one best action, only that best action needs to

be sampled. Despite this, external sampling employs the same sampling approach at all nodes. By adjusting sampling strategy according the the potential value of information to be gained, a far superior sampling strategy can emerge.

Unfortunately, the precise distribution of the value of information in the game tree is unknown. To remedy this, we can look toward an idea prevalent in statistics known as *importance sampling*. Importance sampling is a method for estimating the properties of a distribution by only being able to sample from a different distribution. In our setting, we can use our already collected samples that are aggregated in $R(I, a)$, the cumulative regret for an information set/action pair, in an analogous manner to better estimate the distribution of the value of information within the game tree.

To achieve this, I propose a mixed sampling method that performs external sampling normally, though it performs outcome sampling whenever two conditions are met:

1. There is a single action with an relatively high probability of being selected according to the strategy.

2. The strategy is reasonably confident that it is correct about that action being the best.

Because the action sequences at such states likely hold little valuable information (as the best action has already been identified), this method may allow the exploitative property of outcome sampling to allow us to selectively traverse game states with high value of information.

To implement this, at each game state, I determined if there was a single best action by referencing the average strategy for that iteration and checking if the probability of selecting any action at that information set exceeded a threshold value (e.g. .85). Because some probabilities (those that correspond to simple rational numbers) are especially likely

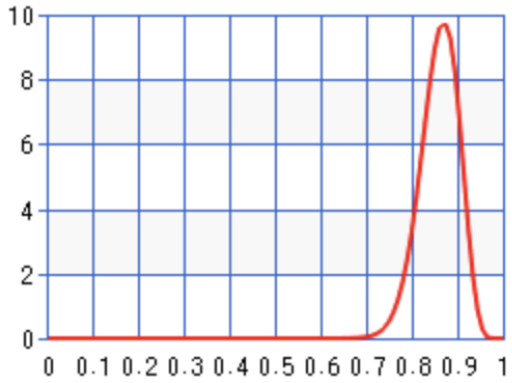


Figure 5.3: Beta(60, 10), an example probability distribution from which cutoff values are sampled in this mixed-sampling strategy. This is the distribution used in the experiements in Figure 5.4. Visualization by Keisan.

to appear in the average strategy, I eliminated the noise caused by small changes in threshold having large effects on results by sampling this threshold value from a beta distribution, a distribution defined on $[0, 1]$. In this way, I was able to vary this cutoff value without noise. The details of this distribution do not need to be known to understand my results, but the pdf of the distribution used to sample for the experiments depicted is seen in Figure 5.3 The distribution has mean .85, though the sampled value may range with moderate probability from .70 to .95.

However, simply identifying if there is a predicted best action is insufficient. I measured the confidence that this single action was in fact the true best through varying methods:

- Measuring confidence by iteration, borrowing the exponential schemes from Section 5.2 in which the lack of confidence decreases exponentially over time.
- Measuring confidence by node, establishing a threshold number of times a node must be visited before outcome sampling can confidently be applied.

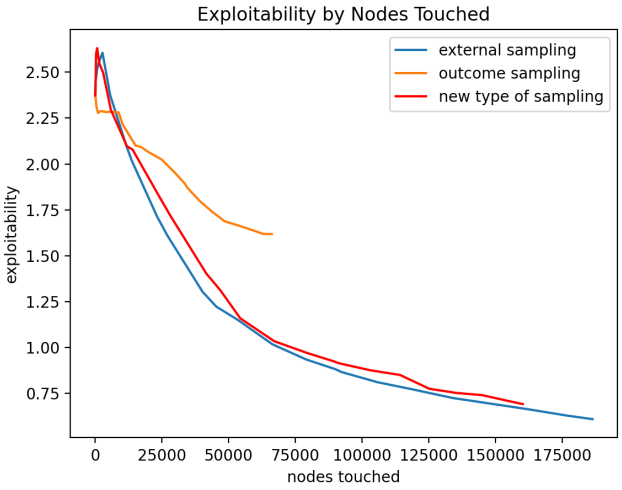
To demonstrate the effects of varying these factors, I present four experiments. In each experiment, the cutoff is sampled from Beta(60, 10) while the method for determining confidence was varied:

- **Pre-set iteration threshold:** A threshold equal to some fraction of the total number of iterations was defined. Before that iteration, external sampling is used. Afterwards, we consider ourselves *confident* in the approximated optimal strategy, and begin to employ outcome sampling where appropriate (based on the cutoff).

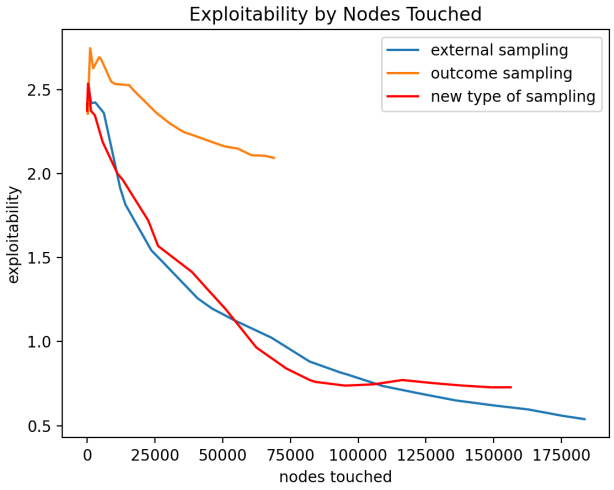
- **Exponentially-decreasing iteration threshold:** This scheme is borrowed from Section 5.2. A uniformly random number x in range $[0, 1]$ is generated. Letting i be the current iteration and I the total number of iterations, if $x < 2^{\left(-\frac{i}{H}\right)}$, for a pre-defined halflife H that does not vary in the number of iterations, we employ external sampling. Otherwise, we consider ourselves *confident* and employ outcome sampling where appropriate that iteration (based on cutoff).

- **Node visit count threshold:** Rather than depending on iteration number, this strategy measures confidence at a node by considering the number of times that node has been visited during training. If this number exceeds a predefined threshold, then outcome sampling is employed where appropriate. Because nodes near the root of the tree are much more likely to be visited than nodes near the leaves, this method causes outcome sampling to be applied more frequently near the root of the tree.

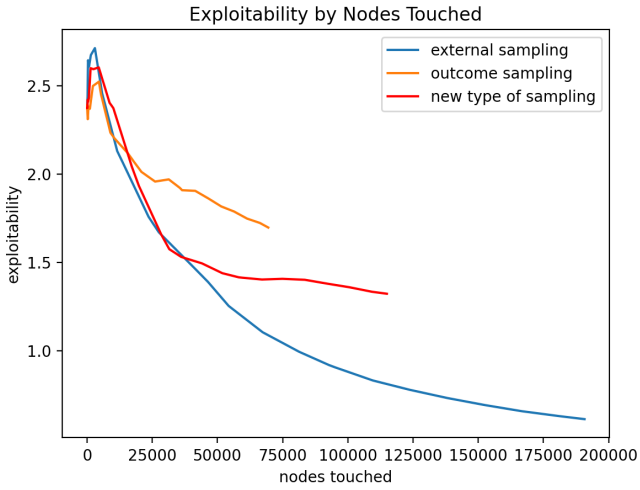
The results of the experiments testing each of these methods can be seen in Figure 5.4. I additionally tested using the most recent strategy rather than the average strategy, though performance was far degraded using the most recent strategy and is not included in the figure.



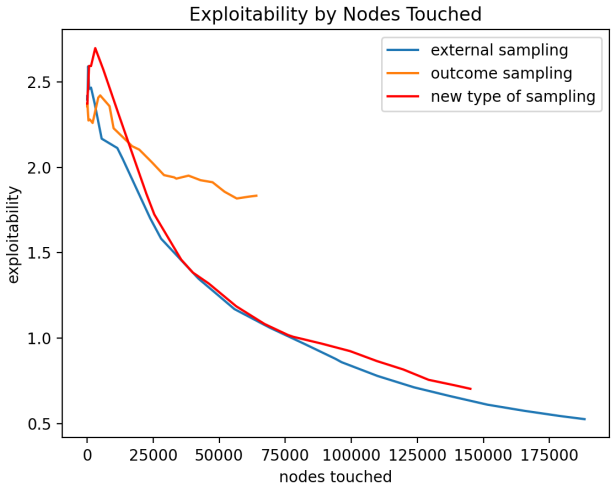
(a) Pre-set iteration threshold of $\frac{I}{2}$



(b) Exponentially decreasing iteration threshold with halflife $2^{10} = 1024$.



(c) Exponentially decreasing iteration threshold with half life $2^3 = 8$



(d) Confidence measured by visits per node, threshold set at 100.

Figure 5.4: This is a sister figure to Figure 5.2. It presents four variations of a mixed sampling strategy in which either ES or OS is selected to be used at each node. In all trials, a cutoff was drawn from Beta(60, 10). In each trial, if a confidence criteria was met, any node with an action probability above the cutoff was traversed with outcome sampling. All trials took place over 2^12 iterations on Leduc Hold'em. 2^{12} iterations performed on Leduc Hold'em with different mixed sampling strategies varying by locus in tree.

In many ways, these results mimic the results from Figure 5.2. Performance for the linear case and per-node case approximately match the performance of external sampling. However, looking at the exponential sampling plots, it is evident that we do not fully remove the exponential factor as we did when applying the exponential scheme to iteration alone.

Why is the exponential speed-up not observed in this case? The optimal strategy for imperfect information games is defined as a probability distribution over the available actions, and so this optimal strategy often does not have a single best action at a given information set. It turns out that for many games, it is rare for a single action to be chosen at any information set with a high probability.

For the game of Leduc Hold'em, I analyzed this pattern by looking at the returned strategy after 2^{16} iterations of training by external sampling. In the returned average strategy, only 23.7% of game states had an action that would be selected with probability $> .85$. So, because external sampling will be applied at all nodes below this cutoff, this mixed sampling strategy is still dependent on external sampling and does not offer the same computational complexity improvements.

This poses two major problems:

1. We do not experience an exponential decrease in time complexity when performing an exponential decrease for confidence, as $> 75\%$ of nodes (for this game) will always only be traversed by external sampling in this method. This can be seen in Figure 5.4c, where despite a very short half life, this mixed sampling strategy visits far more nodes than pure OS. So, this sampling strategy is not generalizable to large games, which is a primary aim.

2. Additionally, because different games vary in how probabilistic the optimal strategy is (the optimal strategy of some games may have zero actions selected with high probability), this sampling method

is contingent on properties of the tree shape, which limits the generality of such a method.

While these downsides limit the applicability of such a sampling method to large games, I am still optimistic about further optimizations to this sampling paradigm, as will be discussed in the next chapter.

5.3.1 Section Review

In this section, I introduced a sampling algorithm that varies by locus in the game tree by incorporating outcome sampling in areas where a single action is thought to be best. However, for the two reasons mentioned, this method does not offer the same generality as the former iteration-varying method. Despite this, this approach was still able to match ES performance on poker-style games, and the framework of varying by locus may offer promising future results.

5.4 Chapter Summary

At the time of the writing of this thesis, among all CFR variants currently developed, the best-performing sampling algorithm for use on poker-style games was external sampling, and the only feasible sampling algorithm on large games was outcome sampling.

In this chapter, we discussed two novel sampling methods that are mixed sampling strategies of external and outcome sampling.

The first method, varying strategy by iteration number, works by mapping iteration numbers to the probability of employing external sampling with a nonincreasing function f. When f is linearly decreasing, performance matches that of standard external sampling, despite half of all iterations using outcome sampling. When f is exponentially decreasing, the computational complexity of this sampling method is

linearly bound in the depth of the tree, allowing for application to extremely large game trees with predicted correspondingly large gains in performance over the current standard of outcome sampling.

The second method, varying strategy by locus in the game tree, works by using the stored cumulative regrets to approximate the value of the information to be gained at different points in the game tree. External sampling is used when this potential value is high, and outcome sampling is used when it is low. This method is able to match the performance of external sampling on poker-style games. Unfortunately, this sampling strategy does not generalize well for imperfect information games with probabilistic strategies, and because this problem can prevent this method from being bound in linear time, it is also less applicable to large games.

6 Moving Beyond Poker

So far, we have demonstrated the effect of mixed-strategy sampling techniques on poker-style games. However, the entire purpose of introducing these methods is to allow them to be applied more generally. Two particular applications of interest are extremely large games or games with high branching factors, and Deep CFR.

6.1 Large Games

In this section, I will offer a preliminary result confirming that a mixed sampling strategy that decreases the probability of selecting ES exponentially with iteration offers a significant performance increase on large games, thus serving as a sampling method than can be applied generally to different tree shapes.

To model extremely large games, I use the model game Liar's Dice. Liar's Dice has a branching factor that increases with the number of dice used. When five six-sided dice are used, this branching factor is 61, which is dramatically larger than that of most poker variants (2 or 3).

The approximate size of a game tree can be given by $\sum_1^d b^d$ (think of summing the number of nodes on each layer of the tree), where b is the average branching factor and d is the average depth. So, game tree size increases exponentially with depth, and polynomially with branching factor (however, the degree of this polynomial tends to be quite large). Why do we say external sampling is infeasible for high branching fac-

tor games rather than large depth games, if depth contributes more to traversed area size?

The answer is that the tree must always be traversed to full depth in CFR, as the regrets are updated according to the utility function stored at each terminal history. So, no sampling method can avoid the need to traverse the tree to full depth, including outcome sampling. However, notice that when $b = 1$ (as it does for the traversed area of outcome sampling), the number of nodes traversed $\sum_1^d b^d$, is equal to d, forming a linear relationship. It is by increasing the branching factor that this exponential increase is felt, and this is why external sampling fails on games with large branching factors. Letting I be the total number of iterations, d the average tree depth, and b the average branching factor, the total number of nodes visited by OS during a training session can be approximated by $I \times d$. Meanwhile, The total number of nodes visited by ES during training can be approximated by $I \times \sum_1^d b^{\lfloor d/2 \rfloor}$ for 2 players, where $\lfloor x \rfloor$ is the floor function.

Because sufficiently large game trees are unable to be traversed by ES in any reasonable amount of compute time, OS will always outperform ES (trivially) for these games. It would still, however, be useful to compare ES performance to both OS performance and that of the novel sampling methods introduced in this section. We have been using exploitability as a metric to measure performance in these games. Unfortunately, calculating exploitability is extremely computationally expensive (it requires traversing the entire tree), and so games that are impractical to traverse with ES are also impractical to have the exploitabilities of their strategies calculated. Even worse, Because ES still samples a single action at every other layer, calculating exploitability requires traversing a number of nodes that grows exponentially greater in tree depth than simply performing an ES traversal. So, for any game that is

large enough for ES to be impractical, it is exponentially *more* impractical to calculate exploitability.

To illustrate this problem, in a game of Liar's Dice with a single 10-sided die, traversing the tree with OS is virtually instantaneous. Traversing with ES requires generally <1 second. However, the time required to compute exploitability exceeded 16 hours (on my machine). As a result, comparing performance as measured by exploitability on these games is unfortunately not a practical task.

A primary result of this thesis is that the sampling method introduced in Subsection 5.2.3, $\mathcal{M}(f)$, where f is exponentially decreasing, bounds computational complexity linearly in the depth of the game tree, rather than exponentially as does ES. As a result, this sampling method theoretically dramatically improves on the current best option for large trees, OS.

Although testing this directly is not possible due to the cost of computing exploitability, it is possible to create a toy approximation of this effect. Specifically, we can use games with a moderate branching factor (so exploitability is still computable), while adding an artificial cost to each iteration performed with external sampling. This is an extremely coarse approximation, but can at least provide an intuition about the type of benefit the exponentially-decreasing ES/OS ratio sampling method will provide in large games where ES is impractical.

To model this, I used a representation of liars dice with a single 6-sided die, which has has a far more humble average branching factor of 13 and average depth 12.41. To model the cost of ES, I made each node traversed via an ES method (in both pure ES and the mixed sampling method) incur ten times the normal cost (that is, the single node counts as ten nodes touched). Due to the high cost of computing exploitability, I computed exploitability at four points throughout training, and additionally bounded training time by the number of nodes touched.

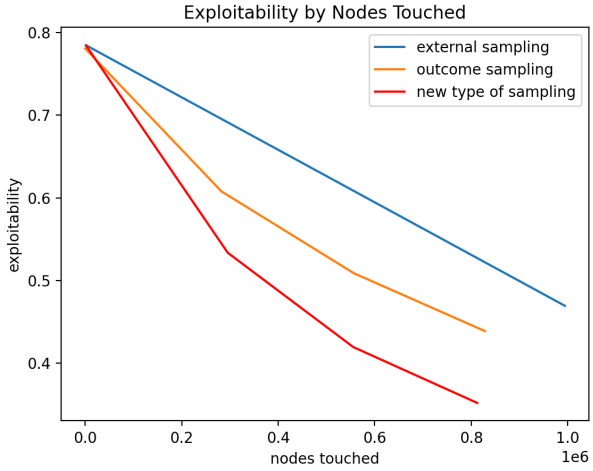


Figure 6.1: The exploitability of three sampling methods over 2^{16} iterations of Liar's Dice. The mixed sampling method uses the proposed exponential decrease in probability of selecting ES with halflife $2^4 = 16$. Despite having the same time complexity as outcome sampling, this mixed strategy shows marked improvement.

Because the halflife should not grow with the number of iterations, the experiment is conducted over 2^{16} iterations of outcome sampling with a very relatively small halflife of $2^4 = 16$ for the mixed sampling method. This corresponds to $Per(\text{ES}) = .00034$ (the percent of iterations for which ES is used), which is a remarkably small fraction. The results can be seen in Figure 6.1.

These results (to be taken with a grain of salt due to the coarseness of the approximation) offer quite a hopeful sign, confirming the results presented in Section 5.2. By using this novel exponentially decreasing mixed sampling strategy, large games impractical for solving with external sampling can show dramatic improvements in performance over the current approach of outcome sampling.

6.2 Deep CFR

Due to the position of Deep CFR as one of the most promising tools developed in the past few years leading toward truly general CFR, I want to briefly discuss how the general improvements to CFR sampling discussed in this thesis could potentially be carried over to Deep CFR to increase its applicability.

Firstly, as discussed in Section 4.3, Deep CFR must collect samples of the game tree to run, and this collection has traditionally been done via external sampling. Because not even Deep CFR avoids a dependency on tree-traversal, it is limited to the same subset of games as its sampling method. So, the introduction of the mixed-sampling method proposed in Section 5.2 already substantially improves the generality of Deep CFR and its performance on large or non-standardly shaped games. Because Deep CFR does not require domain-specific knowledge or game tree abstraction, the introduction of this mixed sampling method alone offers significant expansion of the game space these general methods can be applied to. However, the potential for similar techniques to further improve Deep CFR extends further.

The core feature of Deep CFR is the introduction of a value net, a neural network for approximating $R^t(I,a)$, the cumulative regret values. The time cost of retraining this neural network at every iteration is quite large, and this is a factor that limits how well Deep CFR can be applied to many types of problems. To address this problem directly, I propose varying the decision to retrain the neural net with iteration, just as sampling strategy was varied with iteration in Section 5.2. In this comparison, retraining the value net is a binary decision analogous to deciding to use external sampling for the iteration. With this framework, retraining the neural net is a tool to gain confidence in the value of $R(I,a)$ that it is approximating. As more iterations are completed, confidence in this approximation grows, and so retraining is less necessary; instead,

we can exploit the high probability that $R^t(I,a) \approx R^{t+1}(I,a)$ to avoid the need to retrain.

On the iterations where retraining is skipped, there are a few options. It is possible to leave the value net untouched, and approximate $R^t(I,a)$ as $R^{t-1}(I,a)$. However, it is also possible to make small adjustments to the network from the previous iteration rather than retraining entirely. For instance, rather than retraining from scratch, one could train the network starting from its existing weights from the previous iteration, introducing a small constant number of new samples to adjust the weights slightly.

Deep CFR stands strongly to benefit from this paradigm of performing actions with probabilities that vary with the value of information, and exploring how this proposed framework can be applied to Deep CFR is a very promising area of future work.

6.3 Further Work

While large games and Deep CFR are two examples of promising applications of the ideas presented in this thesis, there are many future directions where this work could be taken.

For example, while this thesis focused on creating more general algorithms with the aim of moving closer to solving complex systems like negotiations, it is also necessary to represent these systems as mathematical objects that can be algorithmically traversed. While there has been some work in mathematically representing these situations as games (Lewis et al. 2017), the work is largely unfinished. For example, developing methods to translate between natural language and a graphical representation of the game could be critical for this goal.

Additionally, it may be possible to develop variants of CFR that can parse patterns in the game tree without needing to traverse the entire

tree. Many games, for instance, are strikingly symmetric. However, CFR is unable to extract any patterns in game tree shape, making simple games difficult to learn. Consider, for example, the game of battleship on an $n \times n$ grid with a single ship of length n. For a reasonable human player, an opponent's ship can be sunk in at most n moves by never selecting the same row or column twice. However, CFR is currently unable to reason about game structure at all.

While there are many directions that can lead toward more general CFR, I hope to highlight one in particular as especially promising: developing more rigorous methods of approximating the distribution of the value of information.

Both sampling variants I proposed exploit the fact that not all game states at all points in time hold information that is equally valuable. Instead, the most valuable information is held by a small group of game states at particular times. An optimal sampling algorithm should selectively visit only the game states with the most valuable information to be gained. In my proposed sampling algorithms, I called attention to patterns, such as information being more valuable in early iterations, and then generated heuristics to exploit these patterns.

However, there have been multiple methods developed that approach the identification of these distributions with more rigor than identifying patterns. Notably, Russell and Wefald have outlined a paradigm of rational meta-reasoning that is able to estimate the value of information to be gained at any node by calculating how large the change in expected value of that node must be to cause a change in strategy much higher up in the game tree (Russell and Wefald 1989). While this technique has been broadly applied in perfect information settings, it has not yet seen any imperfect information use I am aware of. There are definitely complications with this approach, as the standard method of traversing the entire tree to perform these calculations is not realistic. However, if these cal-

culations can be coupled with sampled traversals, we could potentially have much more fine-tuned granularity with which we can identify the value of information.

6.4 In Review

The aim of this thesis as a whole has been to offer a path forward on the matter of how to best increase the generality of how equilibria are computed in imperfect-information zero-sum games.

In pursuit of this goal, I defined CFR as a promising algorithm for this purpose, and presented two major classes of CFR sampling algorithms, a mixed sampling method that varies with iteration, and a mixed sampling method that varies with locus in the game tree. Overall, while both methods show promise as areas of future work, I present the first method (varying by iteration) as a general sampling method that not only extends CFR's applicability to large games, far outperforming outcome sampling, but also matches the performance of the current best methods on poker-style games as well.

While these sampling algorithms provide a concrete step toward more general CFR, I also want to highlight the framework that lead to these algorithms in the first place. Both algorithms rely on the nonuniform distribution of value of information across game states and iteration number in the training process, and these algorithms yield the performance they do because they are able to selectively traverse the areas of the game tree with the highest density of importance in information. This same principle, distributing computation according to the value of information, can be applied in numerous other settings as part of more general solutions.

As a closing note, I hope to emphasize that the work in this area is far from done and there is enormous potential for CFR and related

algorithms to be generalized further. As a component of this ongoing effort, I hope for this thesis to chart out one particular promising path among many.

References

Brown, N., A. Lerer, S. Gross, and T. Sandholm (2019). "Deep counterfactual regret minimization". In: *International conference on machine learning*. PMLR, pp. 793–802.

Brown, N. and T. Sandholm (2019a). "Solving imperfect-information games via discounted regret minimization". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01, pp. 1829–1836.

— (2019b). "Superhuman AI for multiplayer poker". *Science* 365:6456, pp. 885–890.

Hagberg, A., D. Schult, and M. Renieris (2021). *PyGraphviz*. https://https://pygraphviz.github.io/.

Hart, S. and A. Mas-Colell (2000). "A simple adaptive procedure leading to correlated equilibrium". *Econometrica* 68:5, pp. 1127–1150.

Heinrich, J. and D. Silver (2016). "Deep reinforcement learning from self-play in imperfect-information games". *arXiv preprint arXiv:1603.01121*.

Johanson, M. (2013). "Measuring the size of large no-limit poker games". *arXiv preprint arXiv:1302.7008*.

Kroer, C., K. Waugh, F. Kilinç-Karzan, and T. Sandholm (2015). "Faster First-Order Methods for Extensive-Form Game Solving". In: *Proceedings of the Sixteenth ACM Conference on Economics and Computation*. EC '15. Association for Computing Machinery, Portland, Oregon, USA, pp. 817–834.

Kuhn, H. W. and A. W. Tucker (1953). *Contributions to the Theory of Games*. Vol. 2. Princeton University Press.

Lanctot, M., E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, et al. (2019). "OpenSpiel: A framework for reinforcement learning in games". *arXiv preprint arXiv:1908.09453*.

Lanctot, M., K. Waugh, M. Zinkevich, and M. Bowling (2009). "Monte Carlo sampling for regret minimization in extensive games". *Advances in neural information processing systems* 22, pp. 1078–1086.

Lewis, M., D. Yarats, Y. N. Dauphin, D. Parikh, and D. Batra (2017). "Deal or no deal? end-to-end learning for negotiation dialogues". *arXiv preprint arXiv:1706.05125*.

Li, H., K. Hu, Z. Ge, T. Jiang, Y. Qi, and L. Song (2018). "Double neural counterfactual regret minimization". *arXiv preprint arXiv:1812.10607*.

Moravčík, M., M. Schmid, N. Burch, V. Lis, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling (2017). "Deepstack: expert-level artificial intelligence in no-limit poker". *arXiv preprint arXiv:1701.01724*.

Nash, J. (1951). "Non-Cooperative Games". *Annals of Mathematics* 54:2, pp. 286–295.

Russell, S. J. and E. Wefald (1989). "On Optimal Game-Tree Search using Rational Meta-Reasoning." In: *IJCAI*. Citeseer, pp. 334–340.

Schmid, M., N. Burch, M. Lanctot, M. Moravčík, R. Kadlec, and M. Bowling (2019). "Variance reduction in monte carlo counterfactual regret minimization (VR-MCCFR) for extensive form games using baselines". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01, pp. 2157–2164.

Steinberger, E., A. Lerer, and N. Brown (2020). "DREAM: Deep regret minimization with advantage baselines and model-free learning". *arXiv preprint arXiv:2006.10410*.

Tromp, J. (2021). *Chess Position Ranking*. https://github.com/tromp/ChessPositionRanking.

Von Neumann, J. (1944). *Theory of games and economic behavior*. eng. Princeton university press, Princeton.

Zinkevich, M., M. Johanson, M. Bowling, and C. Piccione (2007). "Regret minimization in games with incomplete information". *Advances in neural information processing systems* 20, pp. 1729–1736.