Computers are incredibly fast, accurate and stupid. Human beings are incredibly slow, inaccurate and brilliant. Together they are powerful beyond imagination.

– Albert Einstein

University of Alberta

MONTE CARLO SAMPLING AND REGRET MINIMIZATION FOR EQUILIBRIUM COMPUTATION AND DECISION-MAKING IN LARGE EXTENSIVE FORM GAMES

by

Marc Lanctot

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

Doctor of Philosophy

Department of Computing Science

Spring 2013
Edmonton, Alberta

Abstract

In this thesis, we investigate the problem of decision-making in large two-player zero-sum games using Monte Carlo sampling and regret minimization methods. We demonstrate four major contributions. The first is Monte Carlo Counterfactual Regret Minimization (MCCFR): a generic family of sample-based algorithms that compute near-optimal equilibrium strategies. Secondly, we develop a theory for applying counterfactual regret minimization to a generic subset of imperfect recall games as well as a lossy abstraction mechanism for reducing the size of very large games. Thirdly, we describe Monte Carlo Minimax Search (MCMS): an adversarial search algorithm based on *-Minimax that uses sparse sampling. We then present variance reduction techniques that can be used in these settings, with a focused application to Monte Carlo Tree Search (MCTS). We thoroughly evaluate our algorithms in practice using several different domains and sampling strategies.

Acknowledgements

The first person I would like to thank is my supervisor Michael Bowling. His guidance, support, and knowledge helped immensely in my pursuit of this degree. I learned a lot from working with him, not just about game theory, but also the value of rigor, persistence, and patience. There were times that I would start a meeting discouraged by recent results, and end the meeting excited by future results. This enthusiasm always propelled me forward and played an important role in my success.

I would like to thank my colleagues at Computing Science and members of the Computer Poker Research Group. In particular, I'd like to thank Michael Johanson, who helped greatly in my quest to understand counterfactual regret minimization, restricted Nash responses, and public-chance sampling. Even when Mike was busy, he always found time to write back with detailed and thoughtful answers to my questions. I would also like to thank Neil Burch and Richard Gibson for introducing me to new sampling techniques, suggesting improvements for the current techniques, and for their important contributions to CFR in the imperfect recall setting. I would like to thank Joel Veness, who was a delight to work with, and whose original ideas inspired the last two chapters of this thesis.

A number of other people helped me along the way, and I would like to thank them for this help as well: Michael Buro, Duane Szafron, Csaba Szepesvári, Martin Müller, Michael Wellman, Martin Zinkevich, Kevin Waugh, Nolan Bard, Abdallah Saffidine, Chris Archibald, Marc Ponsen, Steve de Jong, Nathan Sturtevant, Volodymyr Mnih, Craig Boutilier, Marc Gendron-Bellemare, Brian Tanner, and Adam White. Working at University of Alberta allowed me to collaborate with many people, and for that I am thankful.

Finally I would like to thank Sheri Bennett, who always believed in me. Her encouragement and support were nice at times, and critically important at others.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The main goal of research in artificial intelligence (AI) is to design algorithms that make intelligent decisions. These algorithms could be used in automated navigation systems for autonomous vehicles, robotic household aids, and computerized opponents in virtual environments. In each of these examples, a decision-making algorithm is faced with several options due to the dynamics of the environment *and* actions of the other *agents* (decision makers). In general, other agents may have goals that conflict with the goal of a decision-making algorithm. For example, two autonomous vehicles may want to use the same stretch of road, or an automated opponent may want to prevent the human from completing its goal in a simulated training mission. As a result, decision-making algorithms in these multi-agent settings must account for the goals and intentions of other agents.

The *extensive-form game* formalism is a powerful and rigorous model for sequential, multi-agent decision-making; thus, it is commonly used to describe such settings. Game theory provides mathematical grounds for the notion of an optimal strategy, at least for the large class of *two-player zero-sum games*, which has been the focus of much research in computational game theory. For this class of games the *Nash equilibrium profile* is a set of strategies, one for each player, describing a way to act that guarantees two important properties for both players: i) employing the equilibrium strategy will result in the highest possible payoff assuming the opponent is maximizing their payoff, and ii) the opponent cannot use a different strategy to achieve a higher payoff than they would by using their equilibrium strategy. Unsurprisingly, a significant amount of effort has been put into designing algorithms that compute or approximate these equilibrium strategies efficiently. The problem of finding Nash equilibria in two-player zero-sum games is challenging since the strategic complexity of the game grows quickly as the size of the game increases. Yet, the most interesting games to researchers — and players — are very large.

The class of zero-sum games is made up of all games which are strictly competitive: a player's utility is defined as a negation of the opponent's utility — "one player's loss is the other player's gain". Zero-sum games include classic games that humans have been interested in for many years (*e.g.,* Chess, Checkers, Go) and model adversarial situations where two agents are competing for a shared or often limited resource such as unclaimed land, unallocated memory, unused bandwidth, or money. Studying how players act strategically in these domains gives insight in how to predict the actions of self-interested agents, which is vital in adversarial environments. For example, consider the problem of designing a robot which will track a criminal suspect. The robot may need to follow the suspect to stay within range of them while the suspect wants to get out of range or otherwise hinder the robot. The decisions made in this interaction are where to move and what can be done to hinder each agent. Knowing the opponent's optimal strategy, or even a near-optimal strategy, is clearly advantageous in this setting. Even the ability to characterize their strategy based on certain facts would be useful.

Classical and Modern Research on Games

Programming computers to make rational decisions in large zero-sum games has been a classic interest to the AI community [104; 96]. Results of these studies spawned an exciting line of research in large extensive-form games, some of which were eventually solved [98; 109; 121; 29; 92; 2], and later inspired man versus machine competitions [100; 15; 48]. A large portion of the existing research focuses on *perfect information games*: games where information known by one or more agents is not hidden from those agents' opponents (*e.g.,* Chess, Checkers, and Go). *Imperfect information games* are games where some information may be known to one or more players and hidden from other players, such as a hand of playing cards. Imperfect information adds another level of complexity since agents must base their reasoning on incomplete states of the game. A number of methods to reduce the complexity of the problem have been well-studied in the perfect information setting; unfortunately these methods are not directly applicable to imperfect information games. In fact, until Koller, Megiddo, and von Stengel's sequence-form linear programming method was introduced in the early to mid 1990s [63], equilibria could only be computed on very small imperfect information games.

Poker is an imperfect information game that has become a popular domain for research. Initial studies date back to very simple versions of the game by von Neumann and Morgenstern [111]. Kuhn studied another small version of Poker [66] where each player is

dealt a single card from a 3-card deck and then play one round of betting with a single chip. Kuhn's simplified poker was later extended to a 13-card deck and analyzed by Gordon [41]. The mid to late 90s and early 2000s saw an explosion of interest in computer Poker, much of which originated from the University of Alberta [10; 82; 99; 8; 59; 64; 105]. By 2003, University of Alberta had developed PsOpti1 and PsOpti2, two computer poker players that had been built by computing equilibria in a reduced version of Texas Hold'em Poker using sequence-form linear programming [11]. A recent regret-minimization algorithm called Counterfactual Regret Minimization (CFR) solved versions of Texas Hold'em Poker with up to 10^{12} game states [124], one of the largest imperfect information games solved to date. Fueled by the 2006 AAAI computer Poker competition [85], a group at Carnegie Mellon University used a slightly different approach to compute good strategies to use in large Poker variants [91; 36]. Two man vs. machine Poker competitions, similar to the famous IBM Deep Blue vs. Kasparov Chess match, were held where a computer bot from the University of Alberta played against human experts [86; 114; 87]. Nash equilibrium strategies were employed in the University of Alberta's "Polaris" program using CFR in these matches; at the second man-machine Poker match Polaris defeated human experts.

Limit two-player Texas Hold'em is a large, strategic game which has become a standard challenge problem used to measure the quality of game-theoretic algorithms. But what about other games? In games with larger branching factors at decision nodes, such as Bluff and No-limit Hold'em, the standard CFR variant (chance sampling) does not scale well. Convergence to an acceptable strategy can take a very long time because each iteration requires computing regret values for every action each player could have taken.

In recent years, Monte Carlo algorithms have been used to overcome the issues of scalability to very large domains. One notable success in the study of games has been in computer Go where researchers developed Monte Carlo Tree Search (MCTS) algorithms [31; 22; 17]. Unlike classical search algorithms, MCTS constructs a model of the game tree incrementally over the course of its simulations. An action selection algorithm based on the bandit literature, Upper Confidence bounds for Trees (UCT), is used to balance exploration and exploitation of actions in each node of the tree [61]. MCTS algorithms with enhancements have greatly improved the play of computer Go players [30].

In this thesis, we focus on computing equilibria and decision-making in large games, with perfect and imperfect information, sing Monte Carlo sampling algorithms.

Overview of Contributions

The main contributions of this work are:

- **A generalized family of game-tree sampling counterfactual regret-minimizing algorithms (MCCFR)**. MCCFR converges faster in theory, and often in practice, than the original no-sampling "vanilla" CFR. Our new sampling schemes ensure faster convergence, particularly in games with large branching factors at decisions nodes. MCCFR is described in detail in Chapter 4.

- **A formalization of counterfactual regret minimization for a general subset of imperfect recall games**. The convergence guarantees of the original CFR applied only to perfect recall games. We derive regret bounds and convergence rates for CFR in these types of games. We then show the effects of an abstraction mechanism used to reformulate very large games as much smaller imperfect recall games. This is described in Chapter 5.

- **Monte-Carlo *-Minimax Search (MCMS): a game-tree sampling search algorithm for large, stochastic, perfect information games**. MCMS uses sparse sampling at chance nodes to reduce the complexity of searching all the subtrees and maintains *-Minimax bound information to generate alpha-beta style cutoffs. We give conditions that guarantee proper pruning with high probability and show its performance in practice in three domains. MCMS is described in Chapter 6.

- **Variance Reduction Techniques**. The quality of Monte Carlo algorithms depends directly on the their underlying estimators and sampling strategies. Hence, in Chapter 7, we investigate the application of variance reduction techniques to these estimators and show how they can be applied to MCTS and MCCFR. We present empirical results showing their practicality in these settings.

Finally, we conclude and describe potential future work in Chapter 8.

Glossary

Most of the important terms in this thesis will be in bold when defined. A description of each of these terms is contained in the glossary section, starting on page 128.

Chapter 2

Background and Related Work

In this chapter we start by giving a summary of the game theory background needed for the rest of the thesis. Then we mention and investigate relevant previous work in some detail. The function of this section is to give sufficient background to properly motivate and describe the algorithms in the remaining chapters.

2.1 Fundamental Game Theory

A **normal form game**, also known as a *strategic-form game* is a game where all players choose a strategy to play simultaneously without knowledge of the other players' choices, all strategies are executed and utilities — usually in the form of payoffs — are given to each player depending on the joint strategy chosen by all players.

Hereon we assume games will always contain two players; many of these definitions apply to more than two players without loss of generality. A *bi-matrix game* is a normal form game with two players whose payoffs are represented using a table. Example bi-matrix games are given in Figure 2.1. In a normal form game player 1 is called the *row player* and player 2 is called the *column player*. Each player has three strategies to choose from: Rock, Paper, and Scissors. Each row represents a strategy (choice) of player 1, each column a strategy (choice) of player 2. Rock, Paper, Scissors is a *zero-sum game* because the payoffs for player 2 are the negation of player 1's payoffs; in these cases it is convention to show only player 1's payoffs in the matrix. Otherwise, an entry in the matrix of the form X, Y corresponds to a payoff of X to the row player and a payoff of Y to the column player.

A **pure strategy** in a matrix game is a single choice that a player can make. Denote the player 1's set of pure strategies S_1 and similarly S_2 for player 2. For example, in the game of Rock, Paper, Scissors: $S_1 = S_2 = \{\text{Rock}, \text{Paper}, \text{Scissors}\}$. A **pure Nash equilibrium**

		Player 2		
		R	P	S
Player 1	R	0	−1	+1
	P	+1	0	−1
	S	−1	+1	0

(a)

		Player 2	
		C	D
Player 1	C	−1 , −1	−10 , 0
	D	0 , −10	−5 , −5

(b)

Figure 2.1: (a) The Game of Rock, Paper, Scissors and (b) The Prisoner's Dilemma

in a matrix game is a pair of pure strategies, one for each player, for which each player has no incentive to deviate from the strategy unilaterally. Let us denote $s_1 \in S_1$ a pure strategy for player 1, $s_2 \in S_2$ a pure strategy for player 2, $s = (s_1, s_2)$ a **pure strategy profile**, and $u_i(s) = u_i(s_1, s_2)$ the payoff to player i when both players play s. Formally, we call s a pure Nash equilibrium if and only if

$$\forall s_1' \in S_1, \forall s_2' \in S_2 : u_1(s) \geq u_1(s_1', s_2) \text{ and } u_2(s) \geq u_2(s_1, s_2'). \tag{2.1}$$

In the Prisoner's Dilemma the only Nash equilibrium is (D,D). In (D,C) player 2 would rather have a payoff of -5 than -10, similarly for player 1 in (C,D). In (C,C) both players would rather have a payoff of 0 than -1. This notion of Nash equilibrium is called *pure* because both players choose D deterministically in their strategy. In Rock, Paper, Scissors a pure strategy Nash equilibrium does not exist; using the same logic there is incentive for one player to deviate at every joint strategy set.

A **mixed strategy** is a probability distribution over pure strategies[1]. Denote the set of mixed strategies for player 1 as Σ_1, similarly Σ_2 for player 2, and $\Sigma = \Sigma_1 \times \Sigma_2$. In Rock, Paper, Scissors a **mixed Nash equilibrium** profile $\sigma = (\sigma_1, \sigma_2)$ does exist, where $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$. Here, σ_i is a probability distribution over S_i for which one player has no incentive to deviate unilaterally because their *expected* payoff would be the same or lower if they did. In fact, it is more general to refer to expected payoffs rather than individual payoffs, so by convention $u_i(\sigma)$ will represent an expected payoff. Formally,

$$u_i(\sigma) = u_i(\sigma_1, \sigma_2) = \mathbb{E}_\sigma[u_i(s)] = \sum_{s_1 \in S_1} \sum_{s_2 \in S_2} \sigma_1(s_1)\sigma_2(s_2)u_i(s_1, s_2). \tag{2.2}$$

We also use σ_{-i} to denote the strategies in σ belonging to player i's opponents. To summa-

[1] Any pure strategy is also a mixed strategy with probability 0 on every choice except one, which has probability 1.

rize, σ is a mixed Nash equilibrium if and only if

$$\forall \sigma_i' \in \Sigma_i : u_i(\sigma) \geq u_i(\sigma_i', \sigma_{-i}) \tag{2.3}$$

holds for each player i.

An ϵ-Nash equilibrium σ is a profile where neither player can gain at most ϵ by unilaterally deviating from σ_i. More precisely, $\sigma = (\sigma_1, \sigma_2)$ is an ϵ-Nash equilibrium if and only if

$$\forall \sigma_i' \in \Sigma_i : u_i(\sigma_i', \sigma_{-i}) - u_i(\sigma) \leq \epsilon \tag{2.4}$$

holds for each player i. Note that because of the way ϵ-equilibria are defined, they can be thought of as approximations to Nash equilibria. In fact, the definition of a Nash equilibrium from Equation 2.3 is a special case of Equation 2.4 with $\epsilon = 0$.

Suppose we have a strategy profile $\sigma = (\sigma_i, \sigma_{-i})$. We call a strategy $\sigma_i \in BR(\sigma_{-i})$ a **best response strategy** to σ_{-i} if it achieves the greatest possible payoff against σ_{-i}. Formally, if

$$\forall \sigma_i' \in \Sigma_i : u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma_i', \sigma_{-i}) \tag{2.5}$$

then σ_i is a best response to σ_{-i}. One property of a Nash equilibrium $\sigma = (\sigma_i, \sigma_{-i})$ is that $\forall i : \sigma_i$ is a best response to σ_{-i}.

In a zero-sum game Nash equilibrium strategies are *interchangeable*: if $\sigma = (\sigma_1, \sigma_2)$ is an equilibrium profile and $\sigma' = (\sigma_1', \sigma_2')$ is another equilibrium profile[2] then (σ_1, σ_2') and (σ_1', σ_2) are also both Nash equilibrium profiles. In addition, when players employ an equilibrium profile in zero-sum games the expected payoff to each player is the same for every equilibrium:

$$u_i(\sigma) = u_i(\sigma') = u_i(\sigma_1, \sigma_2') = u_i(\sigma_1', \sigma_2).$$

In a zero-sum game, the *game value* is equal to the expected payoff to the first player when players employ a Nash equilibrium profile σ: $v = u_1(\sigma)$. In other words, when both players play equilibrium strategies, player 1 gets an expected payoff of v and player 2 gets an expected payoff of $-v$. Suppose player 1 makes a slight adjustment to their strategy; the new strategy is σ_1' and, as a result, $\exists \sigma_2' \in \Sigma_2 : u_1(\sigma_1', \sigma_2') = \min_{\sigma_2'' \in \Sigma_2} u_1(\sigma_1', \sigma_2'') = v - \epsilon_1$. Player 1 has become *exploitable* by an amount $\epsilon_1 \geq 0$. Player 2 can change to a strategy that *exploits* this error and achieve $u_2(\sigma_1', \sigma_2') = -v + \epsilon_1$. A similar argument can be made for player 2 becoming exploitable by an amount ϵ_2. Often the **exploitability** value $\epsilon_\sigma = \epsilon_1 + \epsilon_2$ is used to measure the distance of a strategy to an equilibrium. In this work we will focus on two-player, zero-sum games.

[2] In general, games may have more than one Nash equilibrium.

2.1.1 Normal Form Game Solution Techniques

A *solution technique* refers to a method of finding a Nash equilibrium. Here, we will give overviews of some of the classical methods for finding an equilibrium in these games, beginning with pure equilibria.

One straight-forward algorithm to find a pure equilibrium follows from the definition. For each entry in the matrix (r, c) first inspect all the entries in the column to ensure that $\forall c' \in S_2 : u_1(r, c) \geq u_1(r, c')$, and similarly for all the entries in the row $\forall r' \in S_1 : u_2(r, c) \geq u_2(r', c)$. If these two conditions are met then (r, c) must be a pure equilibrium by definition.

Another algorithm for finding pure equilibria is the *iterated elimination of dominated strategies*. A *dominated strategy* for player 1, $s_1 \in S_1$, is one for which $\exists s_1' \in S_1, s_1 \neq s_1', \forall s_2 \in S_2 : u_1(s_1', s_2) > u_1(s_1, s_2)$. Similarly for player 2. Clearly, dominated strategies are never part of an equilibrium because there is incentive to choose s_1' over s_1 for every choice of the opponent. Therefore, s_1 can be safely removed from the game since it will never be considered part of any equilibrium, defining a new reduced game where $S_1' = S_1 - \{s_1\}$. This is repeated alternately for each player until dominated strategies can no longer be removed. If a single entry remains, it is a Nash equilibrium. If many entries remain then they can be inspected individually as stated above.

If no pure equilibria exist then a mixed equilibrium must exist since every game has at least one Nash equilibrium[3]. One way to find a mixed equilibrium is by following a method inspired by the following fact[4]: for an equilibrium profile σ let σ_i represent the strategy used by player i and σ_{-i} the strategy used by the opponent,

1. If two pure strategies $s_i, s_i' \in S_i$ have positive probability in σ_i then $u(s_i, \sigma_{-i}) = u(s_i', \sigma_{-i}) = \max_{s_i'' \in S_i} u(s_i'', \sigma_{-i})$.

2. If a pure strategy $s_i \in S_i$ has positive probability in σ_i and a different pure strategy $s_i' \in S_i$ has zero probability in σ_i then $u(s_i, \sigma_{-i}) \geq u(s_i', \sigma_{-i})$.

We will refer to this fact as the *best response condition* [113].

A mixed strategy can be modeled as a set of variables whose values correspond to the probability that a particular strategy $s_i \in S_i$ is played. Suppose σ_{-i} is modeled in

[3] This is true for all *finite* games; we will not discuss infinite games in this document. This was John Nash's famous result [76]. A modern version of the proof is found in section 3.3.4 of [106].

[4] This fact has been called different things by different people. Gintis calls it "The Fundamental Theorem of Mixed Nash Equilibria" [38]; Von Stengel calls it "the best response condition" [113]; Osborne calls it "Characterization of mixed strategy Nash equilibrium of finite games" [81].

this way. Then, the expected payoff of a particular strategy $s' \in S_i$ played against σ_{-i} can be expressed in terms of these variables. Another constraint can be constructed in the same way using some other strategy $s'' \in S_i, s'' \neq s'$. Equating these two formulas gives an equation in unknown variables. This can be repeated until there are enough equations to solve for the unknown variables analytically. For example, to find a mixed equilibrium in the Rock, Paper, Scissors, assume that the column player's strategy distribution is $(R, P, S) = (\alpha, \beta, 1 - \alpha - \beta)$. Then the row player's expected payoff when playing R would be $-\beta + (1 - \alpha - \beta)$, when playing P would be $\alpha - (1 - \alpha - \beta)$, and when playing S would be $-\alpha + \beta$. Equating these yields a mixed strategy for the row player of $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Repeating the process for the column player gives the same strategy; together these two strategies form a mixed equilibrium profile. However, this method only applies when the *support*[5] of the equilibrium strategies is known. If strategies outside the support are included, then the assumption that the payoffs are all equal cannot be made due to the second part of the best response condition. For our Rock-Paper-Scissors example above, we assumed that the support included all strategies.

For zero-sum games, another method to find a mixed Nash equilibrium is to formulate the problem as a linear program (LP) and solve it using known LP-solving algorithms. In this context, the row player uses row vector strategy $\mathbf{x} = (x_1, \ldots, x_M)$ and the column player uses column vector strategy $\mathbf{y} = (y_1, \ldots, y_N)^T$. Each vector represents a probability distribution over S_i: the entries represent probabilities of choosing one of the strategies $s_i \in S_i$. The expected payoff to player 1 in a zero-sum game is a matrix product that gives a single (scalar) value $z = \mathbf{xAy}$ and to player 2 is $-z$, where $\mathbf{A}$ is the payoff matrix with entries a_{ij}. Assuming that their opponent is playing to maximize their expected utility, the row player wants to maximize z whereas the column player wants to minimize z. Since $\mathbf{x}$ and $\mathbf{y}$ represent probability distributions there are linear constraints on their values: for the row player $0 \leq x_i \leq 1$, and $\sum x_i = 1$. Similarly for the column player. One can obtain a mixed Nash equilibria by solving an optimization problem whose objective function is

$$\max_{\mathbf{x} \in \Sigma_1} \min_{\mathbf{y} \in \Sigma_2} \mathbf{xAy}. \tag{2.6}$$

The best response condition states that $\mathbf{y}$ is a mixed best response to $\mathbf{x}$ if and only if each strategy in the set $\{s_i \in S_i : y_i > 0\}$ is also a pure best response to $\mathbf{x}$. This condition allows the LP to minimize over player 2's pure strategies. The condition also allows the removal of the $\min_{\mathbf{y}}$ from the LP objective function as it can be replaced by a number of

[5]The support of an equilibrium strategy is the set of strategies to which positive probability is assigned.

$$\begin{aligned}
&\text{maximize} \quad \min_{j \in \{1,\ldots,N\}} \sum_{i=1}^{M} a_{ij} x_i \\
&\text{such that} \quad \sum_{i=1}^{M} x_i = 1 \\
&\qquad x_i \geq 0 \text{ for all } i \in \{1, \ldots M\}
\end{aligned}
\qquad\qquad
\begin{aligned}
&\text{maximize} \quad z \\
&\text{such that} \quad z \leq \sum_{i=1}^{M} a_{ij} x_i \text{ for all } j \in \{1, \ldots, N\} \\
&\qquad \sum_{i=1}^{M} x_i = 1 \\
&\qquad x_i \geq 0 \text{ for all } i \in \{1, \ldots, M\}
\end{aligned}$$

Figure 2.2: The original optimization problem (left) and linear program (right) constructed for player 1 to solve two-player, zero-sum, normal form games using the original objective function (left) and the modified objective function (right). Here, player 1 has $|S_1| = M$ strategies, player 2 has $|S_2| = N$ strategies, and the payoff matrix $\mathbf{A}$, whose entries are a_{ij} has M rows and N columns.

constraints over all strategies $s_2 \in S_2$; the value of $\mathbf{xAy}$ will be the same if $\mathbf{y}$ is mixed or pure. The formulation of the optimization problem and resulting linear program for player 1 are shown in Figure 2.2.

2.1.2 Extensive Form Games and Classical Solution Techniques

We begin our description of extensive form games with elements common to both perfect information and imperfect information games. We will then define the elements specific to imperfect information games.

An extensive form game represents a game with sequences of actions taken in turn rather than two single actions taken simultaneously. Extensive games represent the sequential interaction of players' choices: each turn, a single player chooses an action, and as a result the game enters a new state. This interaction is structured as a game tree. A node in the tree represents either a chance event (a die roll, hand deal, etc.) or a state of the game (whose turn it is to play, what information is available to the player, etc.) while an arc represents the outcome of a chance event or the effect of taking an action. The root node represents the start of the game. Each leaf node represents the end of a game.

Denote $N = \{1, 2\}$ as the set of players. Denote the finite non-empty set of actions that player i can choose $\mathcal{A}_i$, and each individual action as $a \in \mathcal{A}_i$. A **history** is a particular sequence of actions that may occur in the game, starting from the root of the tree. Denote the set of histories H and histories themselves $h \in H$, including a special history $\emptyset$ called the *empty history*. A **terminal history** is a history from root to leaf. Denote the set of

terminal histories $Z \subseteq H$ and individual terminal histories $z \in Z$. A *successor* of a history $h \in H$, written $h' = ha$, is a history that represents the history after having taken a from h. A history h is a **prefix history** of some other history h', written $h \sqsubseteq h'$, if and only if $h' = h$, h' is a successor of h, or there exists a set of intermediate histories $\{h_1, h_2, \ldots, h_n\}$ for $n \geq 1$ such that h_1 is a successor of h, h_{i+1} is a successor of h_i, and h' is a successor of h_n. Every player has a payoff function $u_i : Z \to \mathbb{R}$ that represents the utility given to $i \in \{1, 2\}$ at the end of a game. In two-player, zero-sum games, $\forall z \in Z : u_1(z) = -u_2(z)$.

In a two-player, zero-sum extensive game with chance events, define the *player function* $P : H \backslash Z \to N \cup \{c\}$. The player function determines whose turn it is to act at a particular history $h \in H \backslash Z$, *e.g.,* each node in the game tree can be labeled whose turn it is to act at that state, $P(h)$. Here, c is a special player called *chance* that uses a fixed strategy. We define the set of actions available at h as $A(h) = \{a \in \mathcal{A}_i : ha \in H, P(h) = i\}$. Also, $\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$ for all $i \neq j$. We will restrict our discussion to finite games, so N, H, Z are all finite, and each $\mathcal{A}_i$ is finite.

An **information set** is a set of histories where i is to act that cannot be distinguished by i due to information not known by i. Formally, denote $\mathcal{I}_i$ the set of information sets belonging to player i: a partition of $\{h : h \in H, P(h) = i\}$ where $A(h) = A(h')$ whenever h and h' are in the same part. In this thesis, we use a convention $P(I) = i$ to mean $P(h)$ such that $I \in \mathcal{I}_i$ and $h \in I$. We also use $A(I)$ to mean $A(h)$ such that $h \in I$. We also define the set of (information set, action) pairs for one player to be $\mathcal{C}_i = \{(I, a) | I \in \mathcal{I}_i, a \in A(I)\}$. The sets $\mathcal{C}_1$ and $\mathcal{C}_2$ will mostly be used to specify the space complexity of the following algorithms.

To further clarify the notion of information sets, we now give an example. In the extensive form version of the game of Rock, Paper, Scissors player 1 selects their action but does not reveal it to player 2; instead it is written down on a piece of paper and placed face-down. Then player 2 makes their choice, and both players reveal their choices simultaneously. Here, there are two information sets and $|\mathcal{I}_1| = |\mathcal{I}_2| = 1$: one containing the empty history belonging to player 1, and one containing $\{\mathrm{R}, \mathrm{P}, \mathrm{S}\}$ belonging to player 2.

While Rock, Paper, Scissors is a simple example, a player may encounter many information sets while playing an extensive form game. For some history $h \in H$ define $X_i(h) = ((I, a), (I', a'), (I'', a''), \cdots)$ to be the sequence of player i's (information set, action) pairs that were encountered and taken to reach h in the same order as they are encountered and taken in h. A two-player extensive form game has **perfect recall** if $\forall i \in \{1, 2\}, \forall I \in \mathcal{I}_i : h, h' \in I \Rightarrow X_i(h) = X_i(h')$. Intuitively, this means that player

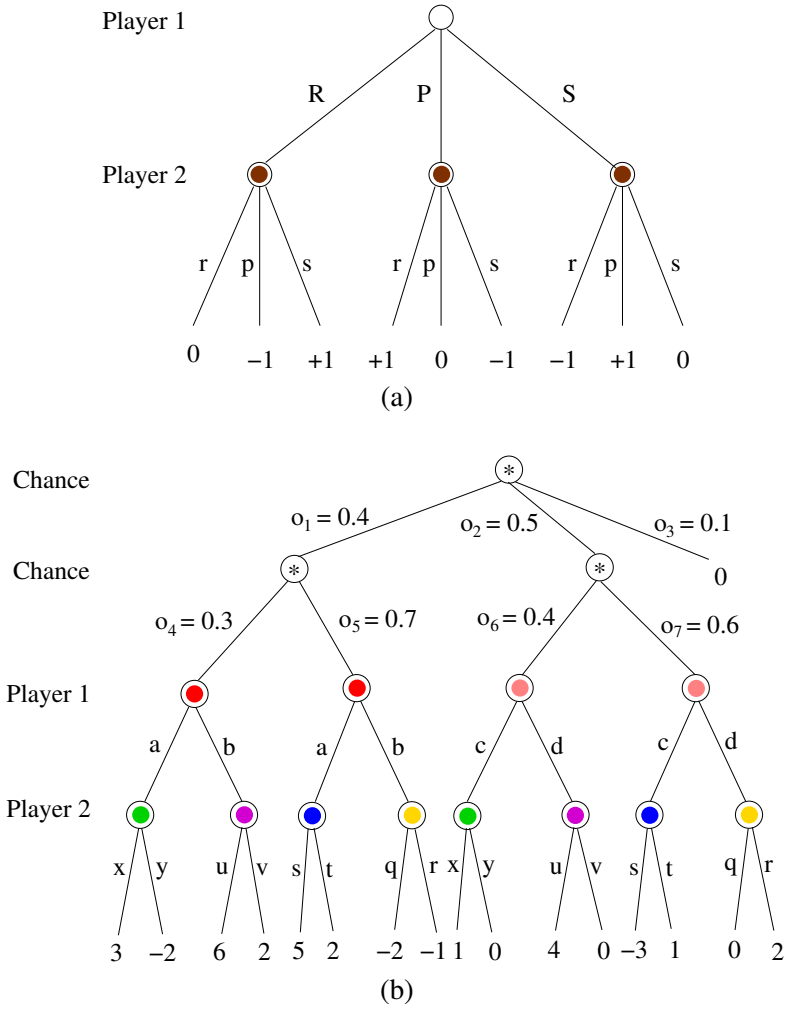


Figure 2.3: (a) The game of Rock, Paper, Scissors in extensive form, and (b) An example of a larger extensive form game. Matching colors denote groupings of nodes that correspond to the same information set. The label $o_i = p$ denotes a chance outcome whose probability is p.

i does not forget any information that they discovered during their play up to h. Unless otherwise noted, and excepting Chapter 5, games are assumed to have perfect recall. Some examples of extensive form games are shown in Figure 2.3.

A **behavioral strategy**, is a function $\sigma_i : \mathcal{I}_i \to \Delta(\mathcal{A}_i)$ that assigns a distribution over actions, one for each information set, where $\Delta(\mathcal{A}_i)$ is the set of probability distributions over $\mathcal{A}_i$. We denote the probability of taking action a in information set $I \in \mathcal{I}_i$ as $\sigma_i(I, a)$; further, we restrict players' choices at I to only those available at I: $\forall i \in N, \forall I \in \mathcal{I}_i, \forall a \notin A(I) : \sigma_i(I, a) = 0$. We denote by $\sigma(I)$ a particular distribution over the action set $A(I)$. When it is obvious from the context, we will drop the subscript i and simply refer to the probability $\sigma(I, a)$. We also use the notation $\sigma(h, a)$ to refer to the probability $\sigma(I, a)$ such that $h \in I$. The chance player, c, is a player that plays with a fixed mixed behavioral strategy σ_c. A **chance event** is a history h such that $P(h) = c$. A **chance event outcome**, or just chance outcome, is a particular action $a \in A(h)$ such that h is a chance event.

Suppose players are using a strategy profile $\sigma = (\sigma_1, \sigma_2)$. Define $I(h)$ to be the information set containing h. The probability of reaching h is defined to be

$$\pi^\sigma(h) = \prod_{h'a \sqsubseteq h} \sigma(h', a).$$

We can split this product into each player's contribution: $\pi_i^\sigma(h)$, $\pi_{-i}^\sigma(h)$, where $\pi^\sigma(h) = \pi_i^\sigma(h)\pi_{-i}^\sigma(h)$. Note, specifically, that $\pi_{-i}^\sigma(h)$ includes chance's action probabilities. Therefore, when $i = 1$, $\pi_i^\sigma(h) = \pi_1^\sigma(h)$ and $\pi_{-i}^\sigma(h) = \pi_2^\sigma(h)\pi_c^\sigma(h)$. Also, for some prefix $h \sqsubseteq z$, let $\pi^\sigma(h, z) = \prod_{h'a \sqsubseteq z, h' \not\sqsupseteq h} \sigma(h', a)$ and define $\pi_i^\sigma(h, z)$ and $\pi_{-i}^\sigma(h, z)$ similarly as before. When both players employ σ, the probability of reaching an information set is defined to be $\pi^\sigma(I) = \sum_{h \in I} \pi^\sigma(h)$. Also, we define $\pi_{-i}^\sigma(I) = \sum_{h \in I} \pi_{-i}^\sigma(h)$. The expected utility of two behavioral strategies can be computed by a game tree traversal that computes the sum

$$u_i(\sigma) = u_i(\sigma_i, \sigma_{-i}) = \sum_{z \in Z} \pi^\sigma(z) u_i(z).$$

Therefore, a Nash equilibrium in behavioral strategies can be defined as before where Σ_i is replaced by the set of behavioral strategies.

Every extensive form game can be converted to a normal form game. A seminal result by Kuhn [66] is that a mixed strategy of this converted normal form game corresponds to a **payoff-equivalent** behavioral strategy in the extensive game it is representing: if σ_i^m is a mixed strategy in the converted normal form game and σ_i^b is a behavioral strategy representation of σ_i^m in the extensive game, and similarly for opponent strategies, then $u_i(\sigma_i^b, \sigma_{-i}^b) = u_i(\sigma_i^m, \sigma_{-i}^m)$. Similarly, any behavioral strategy has a payoff-equivalent

mixed strategy. Therefore, one way to solve an extensive form game is to first convert to normal form, solve the normal form game, and then employ the solution's behavioral representation in the extensive form game. In the converted normal form game, a pure strategy for player i is defined as a function $s_i : \mathcal{I}_i \to \mathcal{A}_i$ such that $\forall I \in \mathcal{I}_i, s_i(I) \in A(I)$. Therefore, there are an exponential number of pure strategies belonging to player i. In particular, if a game has a branching factor of $|A(I)| \geq 2$ for all I, then the number of pure strategies for player i is $O(2^{|\mathcal{I}_i|})$. Hence, the size of the converted normal form is also exponential in $|A(I)|$. Many of the entries in the payoff matrix can be redundant; they are duplicated because many combinations of pure strategies lead to the same leaves in the game tree. There are methods to reduce the normal form game to remove these redundant entries [113]. However, in general, the reduced size may still be exponentially large. As a result, solving an extensive game by solving its converted normal form is impractical on all but the smallest games.

2.1.3 Sequence-Form Linear Programming

Section 2.1.1 described how to solve a zero-sum normal form game by formulating the problem as a linear program. The previous section described how an extensive game can be converted to a normal form but with an exponential increase in the size of the description.

One way to formulate the problem of solving an extensive game as a linear program while avoiding the exponential increase is to impose constraints influenced by the structure of the game tree rather than the normal form's mixed strategies. To achieve this goal, we define and construct *realization plans* [63; 13]. As mentioned above, we have the set of (information set, action) pairs of player i to be $\mathcal{C}_i = \{(I, a) | I \in \mathcal{I}_i, a \in A(I)\}$. Due to perfect recall, $I \in \mathcal{I}_i$ uniquely identifies a sequence of all past information sets belonging to i and actions taken by i before reaching I; let this sequence be $X_i(I) = (c_1, c_2, \ldots, c_n)$, possibly empty, where $c_j \in \mathcal{C}_i$. For $c = (I, a)$ and some behavioral strategy σ_i, let $\sigma_i(c) = \sigma_i(I, a)$ be the probability of taking action a at I. A pure behavioral strategy s *reaches* I if all the actions taken at past information sets lead to I. For a sequence $X_i(I)$ we say that $c_{j'}$ is an ancestor of c_j if $j' < j$ and a *parent* if $j' = j - 1$. A *realization plan* x fulfills

$$x(\emptyset) = 1, \qquad x(I', a') = \sum_{a \in A(I)} x(I, a) \tag{2.7}$$

where (I', a') is a parent of (I, a) or $\emptyset$ if it has no parent. Realization plans and behavioral strategies are interchangeable: to obtain a behavioral strategy from a realization plan simply normalize $\{(a, x(I, a)) | a \in A(I)\}$. To obtain a realization plan from a behavioral strategy,

for each $I \in \mathcal{I}_i$ and $a \in A(I)$ compute $x(I, a) = \sigma_i(I, a)\pi_i^\sigma(h)$ for some $h \in I$.

Given these constraints a new linear program is constructed that can be solved to find a set of weights which correspond to the players' goal of maximizing their payoffs. The new linear program is defined using matrices which represent the structure of the extensive game: $\mathbf{E}$ (structure of the information sets and actions for player 1), $\mathbf{F}$ (structure of the information sets and actions for player 2), and $\mathbf{A}$ (expected payoff per 2-tuple $((I_1, a_1), (I_2, a_2))$ used by each player). In $\mathbf{E}$ and $\mathbf{F}$ there is a row for every information set and an entry in the row for the sequence that leads to the row's information set (with the value 1) and each other sequence for which this sequence is a prefix (with the value -1). $\mathbf{A}$ has an entry for each combination of (I, a) pairs belonging to separate players; this matrix's entries are all zero except for entries $((I_1, a_1), (I_2, a_2))$ that together lead to the end of a game (corresponding to leaves of the tree). Note that there may be multiple terminal histories that satisfy the combination of (I, a) pairs due to chance nodes, which is why the entry represents an expected payoff.

In Section 2.1, we described the problem of finding equilibria as $\max_{\mathbf{x}} \min_{\mathbf{y}} \mathbf{xAy}$, where $\mathbf{x}$ and $\mathbf{y}$ are player strategies and $\mathbf{A}$ is a payoff matrix. Here, the descriptions of the constraints and objective are more complex: $\mathbf{x} \in Q_1$, $\mathbf{y} \in Q_2$ where Q_i is the set of realization plans for player i. So the problem then becomes to find the solution to:

$$\max_{\mathbf{x} \in Q_1} \min_{\mathbf{y} \in Q_2} \mathbf{xAy} = \min_{\mathbf{y} \in Q_2} \max_{\mathbf{x} \in Q_1} \mathbf{xAy} \tag{2.8}$$

with constraints derived from Equation 2.7. The order of the minimizing and maximizing in Equation 2.8 can be swapped due to the minimax theorem [112]. This can be represented as a linear program. Solving the LP using these matrices will give a realization plan for each player, when converted to a behavioral strategies, correspond to Nash equilibrium strategies.

Suppose the linear program has n variables and m constraints. The input size of the LP has mn coefficients for the variables in the constraints, m numbers for the right-hand size of the constraints, and n coefficients for the objective function. The size of the linear program is $L = mn + m + n$. There are many existing algorithms that solve LPs. While the simplex algorithm tends to work well in practice, it can run for $O(2^n)$ iterations. The current best worst-case time complexity for solving LPs is $O(n^{3.5}L)$ [57; 1]. The variables are the $x(I, a)$ of the realization plans from Equation 2.7; there is one for each (I, a) pair of a single player; there can be one per node of the tree in the worst case. Likewise, there will be at least one constraint per information set. Therefore the worst-case time is polynomial in

the size of the game tree. In imperfect information games, the upper bound for n is the sum of the actions available at all their information sets, which can still be prohibitively large. A game with 1 million choices per player could take time proportional to $(10^6)^{3.5} = 10^{21}$ to solve. Therefore, while the sequence form is a dramatic improvement over solving a converted normal form game, the exponent of the polynomial still makes solving large games impractical.

As for the space complexity: in general, the $\mathbf{A}$ matrix will have one non-zero entry per $((I_1, a_1), (I_2, a_2)) \in \mathcal{C}_1 \times \mathcal{C}_2$ that leads directly to a leaf. Equation 2.7 describes an information set tree for a single player i; if we imagine a constant branching factor for both players at all nodes of $|A(I)| = 2$ then the number of leaves in this tree is at least $\frac{|\mathcal{I}_i|+1}{2} \approx \frac{|\mathcal{C}_i|}{4}$. It is possible that every leaf be paired with every leaf of the opponent's information set tree, so the amount of space required to store the $\mathbf{A}$ matrix is $O(|\mathcal{C}_1||\mathcal{C}_2|)$. Storing $\mathbf{E}$ and $\mathbf{F}$ requires $O(|\mathcal{C}_1|)$ and $O(|\mathcal{C}_2|)$, respectively, so the total the amount of space required to store the entire linear program is $O(|\mathcal{C}_1||\mathcal{C}_2| + |\mathcal{C}_1| + |\mathcal{C}_2|)$.

2.2 Iterative Learning and Regret Minimization

The techniques from Sections 2.1.2 and 2.1.3 are classical methods for computing equilibria in two-player, zero-sum extensive form games. These classical techniques determine the equilibrium strategies by solving an optimization problem inspired by von Neumann's minimax theorem [112].

A different approach to computing Nash equilibria is to learn them over time by repeatedly modifying strategies in a specific way while collecting information that is used to compute a series of ϵ-Nash equilibria. The idea is to incrementally improve upon the current solution such that $\epsilon \to 0$ over time, so that the sequence of intermediate solutions is converging to a Nash equilibrium (*i.e.* $\epsilon = 0$).

There are many iterative solution techniques for computing approximate equilibria in extensive form games. Here, we start with a survey of the most recent, relevant techniques but then focus primarily on *counterfactual regret minimization* as it is the most relevant to the construction of the MCCFR family of algorithms presented in Chapter 4.

2.2.1 Excessive Gap Technique for Finding Nash Equilibria

Sequence-form linear programming (see Section 2.1.3) tends to work quite well in practice on games with a few thousand information sets. However, even the best linear programming solvers struggle with games that have millions of information sets. The full optimization

problem described in Section 2.1.3 can be reformulated into a relaxed optimization problem that is easier to solve. The resulting solution may be sub-optimal by some amount (ϵ).

One way to iteratively improve on ϵ-equilibria is to reformalize the optimization problem in Equation 2.8 by perturbing the objective function to make it smooth (differentiable) and convex. Let O be the original optimization problem, $S[O]$ be the smoothed optimization problem, μ^0 be the initial parameters of $S[O]$, and ϵ^0 be the initial suboptimality of a solution to $S[O]$ with respect to the optimal solution to O. Gradient decent along the objective function is used to find a new set of parameters μ^1; these new parameters lead to a new ϵ^1. The process is repeated generating ϵ-equilibria with ϵ decreasing in the limit as the number of iterations tends to infinity. This idea is applied through Nesterov's *excessive gap technique* (EGT) [36; 46].

The key result that makes this method possible is that the difference between the optimal value of the original optimization problem O (Equation 2.8) and the smoothed optimization problem $S[O]$ is bounded when the smoothed functions satisfy an *excessive gap condition*. Recall that that Q_i is the set of realization plans for player i. From Section 2.1.3 we can rewrite Equation 2.8 in a slightly different form:

$$\max_{\mathbf{x}\in Q_1} f(\mathbf{x}) = \min_{\mathbf{y}\in Q_2} \phi(\mathbf{y})$$

where

$$f(\mathbf{x}) = \min_{\mathbf{y}\in Q_2}\{z\}, \qquad \phi(\mathbf{y}) = \max_{\mathbf{x}\in Q_1}\{z\},$$

and

$$z = \mathbf{x}\mathbf{A}\mathbf{y}.$$

If we have strongly convex functions d_i and scaling constants $\mu_1, \mu_2 > 0$ we define smoothed functions:

$$f_{\mu_2}(\mathbf{x}) = \min_{\mathbf{y}\in Q_2}\{z + \mu_2 d_2(\mathbf{x})\}$$
$$\phi_{\mu_1}(\mathbf{y}) = \max_{\mathbf{x}\in Q_1}\{z - \mu_1 d_1(\mathbf{y})\}$$

If these smoothed functions satisfy the excessive gap condition $\forall \mathbf{x} \in Q_1, \forall \mathbf{y} \in Q_2, f_{\mu_2}(\mathbf{x}) \geq \phi_{\mu_1}(\mathbf{y})$, then the difference between the non-smooth functions is bounded as follows: $0 \leq \phi(\mathbf{y}) - f(\mathbf{x}) \leq \mu_1 D_1 + \mu_2 D_2$, where D_i is the maximum value of d_i over all realization plans for i.

The EGT algorithm iteratively generates values $(\mu_1^k, \mu_2^k, \mathbf{x}^k, \mathbf{y}^k)$ for $k = \{0, 1, \cdots\}$ and will compute an ϵ-equilibrium in $O(\frac{1}{\epsilon})$ iterations. Starting with an initial point $(\mathbf{x}^0, \mathbf{y}^0)$ that satisfies the gap condition, each iteration of EGT shrinks one of the μ_i parameters so

that $\mu_i^{k+1} < \mu_i^k$ while preserving the gap condition on $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})$. The shrink procedure depends on the function

$$\text{sargmax}(d_i, \mathbf{Ay}) = \max_{\mathbf{x} \in Q_1} \{\mathbf{x}(\mathbf{Ay}) - d_i(\mathbf{x})\} \tag{2.9}$$

which finds the optimal $\mathbf{x}$ in the smoothed optimization problem if the opponent uses $\mathbf{y}$. The sargmax in Equation 2.9 is defined for player 1 but can be similarly defined for player 2. The gradients of the f_{μ_2} and ϕ_{μ_1} are followed by computing new points and parameters through applications of sargmax.

The computation of sargmax can be efficient because of the following result. There exist certain specialized functions[6], d_i, which make sargmax into a closed-form expression. For example, this is true when using a particular $\mathbf{x}$ from a simplex domain:

$$\Delta_n = \mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq \mathbf{0}, \sum_{x_i \in \mathbf{x}} x_i = 1$$

When the domain is the set of realization plans, Q, then sargmax can be computed by repeatedly applying the closed form functions for simplices [36]. The game tree can be broken down into a set of simplices, one for each information set, and a tree traversal can be used to calculate the optimal values needed for a given realization plan and choice of nice prox function.

2.2.2 Regret Minimization and Games

A multi-round *multi-armed bandit problem* is a situation where an agent must make repeated decisions amongst a fixed number of action K. Each action (pulling of a bandit arm) gives a payoff, possibly drawn from some distribution, but whose mean payoffs are unknown. When the decision-maker decides to choose an action, it receives a single payoff from that arm. The intent is to make a series of decisions which results in the highest total payoff. The problem is an *online learning problem* because the agent only discovers its payoff for making the decision after it has been made. Therefore, agents must decide between *exploring* (trying its actions) and *exploiting* (choosing the action with the highest estimated expected utility).

Suppose some online learning algorithm makes decisions for T steps and observes a sequence of (action, utility) pairs: $((a^1, u^1), (a^2, u^2), \ldots, (a^T, u^T))$ where $i^t \in \{1, \ldots, K\}$ is the decision made at step t, u^t is the utility earned by the decision-maker at step t, and u_a^t is the utility for choosing action a at step t. In general, the algorithm choosing the actions

[6] Referred to as "nice prox functions" by the authors of [36; 46]

and process determining the rewards are stochastic, so we will talk about expected values. Define the expected **regret** of not taking action a at every time step as

$$R^T(a) = \mathbb{E}\left[\sum_{t=1}^{T}(a_i^t - a^t)\right]. \tag{2.10}$$

One way to measure the quality of an online learning algorithm is to measure its accumulated **external regret**:

$$R^T = \max_{i\in\{1,\ldots,N\}} R^T(i) = \max_{i\in\{1,\ldots,K\}} \mathbb{E}\left[\sum_{t=1}^{T}(a_i^t - a^t)\right] \tag{2.11}$$

In other words we measure how much the algorithm regrets not taking the *best single decision in hindsight*. Define the average regret to be $\bar{R}^T = R^T/T$. The general goal is to have low regret. We say that an algorithm is a **regret minimization** algorithm if the algorithm's average regret, approaches 0 over time:

$$\lim_{T\to\infty} \bar{R}^T = \lim_{T\to\infty} \frac{R^T}{T} = 0. \tag{2.12}$$

By convention, the payoffs are often normalized with respect to the maximum range, *i.e.,* to the range [0,1], without loss of generality.

There is a large field of research devoted to finding good ways to make decisions in these environments. Often, statistical assumptions are made about the environment, e.g. that the payoff distribution of each arm is fixed. In general, the payoffs or payoff distributions for each decision are not fixed; they can be chosen by an adversary. Since this situation is more relevant to describing how online learning algorithms apply to game-solving, we choose to focus on it.

Often in the online learning literature, negative payoffs are incurred (*losses*) and rather than maximize total payoff, the goal is to minimize total loss. A loss l_i^t refers to a loss of choosing decision i at time t. One way to minimize regret if the losses l_i^t are binary ($l_i^t \in \{0, 1\}$) and the loss for every option is revealed after each step is to use the Randomized Weighted Majority (RWM) algorithm [75; 13]. A decision problem is repeated, and at each time step the decision-maker chooses a strategy from a distribution where the probability of selecting strategy i is

$$\frac{w_i}{\sum_{j=1}^{N} w_j}, \quad \text{where} \quad w_i = (1-\eta)^{L_i}$$

where η is a small constant and $L_i = \sum_{t=1}^{T} l_i^t$. If η is set to $\min(\sqrt{(\ln K)/T}, 1/2)$, this rule leads to a regret of at most $O(\sqrt{T \ln K})$ [13, Theorem 4.5]. The Hedge algorithm is a generalization of RWM that can be applied when $l_i^t \in [0, 1]$ [28].

The Exp3 algorithm mixes its choice between Hedge and a uniformly random strategy [5]. The RWM and Hedge algorithms require that the losses for each action be revealed at each time step. The Exp3 algorithm circumvents this by defining a simulated reward vector where all payoffs are 0 except the chosen action, and multiplies the payoff it receives by the inverse of the probability of choosing that action. Using this rule, Exp3 can guarantee that the regret is no more than $O(T^{\frac{2}{3}}(N \ln N)^{\frac{1}{3}})$.

There is an important connection between regret minimization and game theory. In constant-sum games, which subsume zero-sum games, if two players play against each other for T time steps and player i has accumulated regret R_i^T, then the *average payoff* for player i is at least $v_i - \frac{R_i^T}{T}$, where v_i is the minimax game value to player i [13]. Therefore, if the strategies are chosen according to any regret-minimizing algorithm, then the average payoff approaches the minimax game value. Define an *average strategy profile* $\bar{\sigma}^T$ in a normal form game to be one where

$$\bar{\sigma_i}^T(s) = \frac{\sum_{t=1}^{T} \sigma_i^t(s)}{T}, s \in S_i \tag{2.13}$$

holds for every player i. The average strategy is defined similarly for behavioral strategies in an extensive form game as

$$\bar{\sigma_i}^T(I, a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I, a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)}, I \in \mathcal{I}_i \tag{2.14}$$

where σ^t is the strategy profile used by all players at step t.

In constant-sum games, if the average regret $\bar{R}^T < \epsilon$ for both players, then the average strategy profile $\bar{\sigma}^T$ is a 2ϵ-equilibrium. This is known as a folk theorem; for details, see [13]. This folk theorem inspires game-theoretic algorithms that can compute Nash equilibria by minimizing regret over repeated plays.

Counterfactual Regret Minimization

The **counterfactual regret** of not playing action $a \in A(I)$ at I is the difference in the expected utility of playing a at I versus playing $\sigma(I)$ at I, weighted by the opponents' probability of reaching I [124; 123]. The counterfactual regret minimization algorithm (CFR) minimizes the maximum counterfactual regret (over all actions) at every information set.

Let Z_I be the subset of terminal histories where a prefix of that history is in I; for $z \in Z_I$ denote $z[I]$ as that prefix. Define *counterfactual value* as:

$$v_i(\sigma, I) = \sum_{z \in Z_I} \pi_{-i}^{\sigma}(z[I])\pi^{\sigma}(z[I], z)u_i(z) \tag{2.15}$$

which is the sum of the expected payoff for having reached I weighted by the opponent's probability of reaching I.

The *immediate counterfactual regret* for a specific information set is defined as:

$$R^T_{i,imm}(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^{T} \left(v_i(\sigma^t_{I \to a}, I) - v_i(\sigma^t, I) \right) \tag{2.16}$$

where $\sigma^t_{I \to a}$ is identical to σ^t except that a is taken at I, The algorithm produces new strategies at each iteration by employing a regret minimizer at each information set over the counterfactual values.

The CFR algorithm, at time step t, computes a regret value:

$$r(I, a) = v_i(\sigma^t_{I \to a}, I) - v_i(\sigma^t, I) \tag{2.17}$$

for each information set I and each action $a \in A(I)$. This regret value represents how much a player i regrets playing σ_i instead of choosing to play $\sigma_{I \to a}$.

In counterfactual regret minimization, regret-matching is applied at each information set to update strategies. Define the *cumulative counterfactual regret* at information set I for action a up to time T as

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

To produce new strategies, **regret-matching** normalizes the positive portions of accumulated regret for each action [43; 42]. More precisely, the strategy that regret-matching produces at I for $a \in A(I)$ is:

$$\sigma^{T+1}(I, a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a' \in A(I)} R_i^{T,+}(I,a')} & \text{if the denominator is positive;} \\ \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases}, \tag{2.18}$$

where $R_i^{T,+}(I, a) = \max\{R_i^T(I, a), 0\}$. Regret-matching yields sublinear regret as a result of satisfying the conditions of Blackwell's approachability theorem [12]. It is tempting to ask why mix over these actions instead of greedily choosing the action with the highest regret? Zinkevich et. al. also showed that minimizing the immediate counterfactual regret also minimizes the average overall regret $\bar{R}^T$. Therefore, the average profile approaches a Nash equilibrium over time in the case of zero-sum games.

The algorithm is summarized in Algorithm 1. We refer to this algorithm as Vanilla CFR when there is no sampling applied, not even at chance nodes.

Algorithm 1 Counterfactual Regret Minimization (Vanilla CFR)

Initialize regret tables: $\forall I, \forall a \in A(I), r_I[a] \leftarrow 0$.
Initialize cumulative strategy tables: $\forall I, \forall a \in A(I), s_I[a] \leftarrow 0$.
Initialize initial profile: $\sigma^1(I, a) \leftarrow 1/|A(I)|$

function CFR(h, i, t, π_1, π_2):
if h is terminal **then**
  return $u_i(h)$
else if h is a chance node **then**
  $\pi'_1 \leftarrow \pi_1$ **if** $i = 1$ **else** $\sigma_c(h, a) \cdot \pi_1$
  $\pi'_2 \leftarrow \pi_2$ **if** $i = 2$ **else** $\sigma_c(h, a) \cdot \pi_2$
  return $\sum_{a \in A(h)} \sigma_c(h, a)$ CFR(ha, i, t, π'_1, π'_2)
end if
Let I be the information set containing h.
$\sigma^t(I) \leftarrow$ RegretMatching(r_I) using Eq. 2.18
$v_\sigma \leftarrow 0$
$v_{\sigma_{I \to a}}[a] \leftarrow 0$ for all $a \in A(I)$
for $a \in A(I)$ **do**
  if $P(h) = 1$ **then**
    $v_{\sigma_{I \to a}}[a] \leftarrow$ CFR(ha, i, t, $\sigma^t(I, a) \cdot \pi_1$, π_2)
  else if $P(h) = 2$ **then**
    $v_{\sigma_{I \to a}}[a] \leftarrow$ CFR(ha, i, t, π_1, $\sigma^t(I, a) \cdot \pi_2$)
  end if
  $v_\sigma \leftarrow v_\sigma + \sigma^t(I, a) \cdot v_{\sigma_{I \to a}}[a]$
end for
if $P(h) = i$ **then**
  for $a \in A(I)$ **do**
    $r_I[a] \leftarrow r_I[a] + \pi_{-i} \cdot (v_{\sigma_{I \to a}}[a] - v_\sigma)$
    $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I, a)$
  end for
end if
return v_σ

function Solve():
for $t = \{1, 2, 3, \cdots, T\}$ **do**
  for $i \in \{1, 2\}$ **do**
    CFR($\emptyset$, i, t, 1, 1)
  end for
end for

Theorem 1 (Theorem 4 of [124]). *When using Vanilla CFR, the average regret for player i is bounded by*

$$\bar{R}_i^T \leq \frac{\Delta_{u,i}|\mathcal{I}_i|\sqrt{|A_i|}}{\sqrt{T}}$$

where $|A_i| = \max_{I \in \mathcal{I}_i} |A(I)|$ *and* $\Delta_{u,i} = \max_{z,z' \in Z}(u_i(z') - u_i(z))$.

Therefore, the algorithm can find an ϵ-Nash equilibrium in $O(\frac{1}{\epsilon^2})$ iterations. Recall the definition of $\mathcal{C}_i$ from Section 2.1.3. Vanilla CFR needs to store a regret and a value for the average strategy for every (I, a) pair for both players. Therefore, the memory required by CFR is $O(|\mathcal{C}_1| + |\mathcal{C}_2|)$, which is an improvement over the product of $|\mathcal{C}_1|$ and $|\mathcal{C}_2|$ required by sequence-form linear programming.

To obtain chance-sampled CFR that is used by most of the Poker implementations, instead of recursively calling CFR for each outcome in lines 9 to 11, a single one is sampled $a \sim \sigma_c(I)$ and the probability $\sigma_c(I, a)$ is not included in π_{-i}. The reason for the latter change will become clear when chance-sampled CFR is described as an MCCFR algorithm in Chapter 4.

In practice, there is a straight-forward pruning optimization that can be applied to this algorithm. Notice that when $\pi_{-i} = 0$, then regret update on line 27 can be skipped. Furthermore, once π_{-i} is set to 0, it will remain so for the remainder of the sub-tree and all the regret updates in the sub-tree can be skipped. The algorithm still needs to traverse the sub-trees for the average strategy update on line 28. If ever both $\pi_{-i} = \pi_i = 0$, then it is safe to prune the remaining sub-trees because both regret and average strategy increments will always be 0. For the remainder of this thesis, we assume that CFR and chance-sampled CFR algorithms always include this pruning optimization.

Online Convex Programming and No-Regret Learning

Convex programming is a generalization of linear programming. The goal is to minimize a convex cost function $c : X \to \mathbb{R}$ where X is some n-dimensional convex set. An *online convex programming* (OCP) problem is an iterative process where at each step the function $c^t(x)$ is not known until a feasible *hypothesis* $x \in X$ is chosen for it, and it can change after every step t. The general goal is to minimize the cost; performance is measured by minimizing the regret of *not* picking a particular choice x.

Generalized Infinitesimal Gradient Ascent (GIGA) is an algorithm that minimizes regret by making adjustments to the strategies (distributions over choices) by following the gradient of the observed cost function [122]. The gradient $\nabla c^t(x)$ is assumed to be given.

A greedy projection algorithm is used to select the next point; essentially, an x is chosen from the feasible set of hypotheses that is closest to $x^t - \eta_t \nabla c^t(x^t)$, where η_t is the learning rate at time t. With an appropriately chosen η_t, the regret bound for this algorithm is shown to be sublinear in the number of steps T, therefore its average regret approaches 0 over time. GIGA can be applied to two-player games by assuming each play is an instance of an OCP, where the cost function is the payoff received for each players' choice of actions. For a particular choice of learning rates, the average regret over time goes to 0, so this algorithm suggests a way to compute the optimal strategies as the average of the empirical play.

A more recent algorithm for solving OCPs is Lagrangian Hedging (LH) [40]. In LH, the state is maintained as an accumulated regret vector, s, updated at each time step. The main idea is to steer this vector s over time towards "safe sets" (s vectors where $\forall x \in X, s \cdot x \leq 0$) by using appropriate *potential functions* $F(s)$ whose gradients are denoted $f(s)$. These potential functions have the property that their value is high when s is far from safe and low when s is close to safe. The next hypothesis x^t is chosen based on $f(s^t)$ at each step, and s^{t+1} is updated to incorporate the regret from the newly observed losses. LH is a generalization of previous online learning algorithms; the choice of potential function is the parameter that differentiates instances. In fact, it generalizes the weighted majority, regret-matching, and Hedge algorithms. As with GIGA, LH can be applied to repeated games in self-play to learn an equilibrium or to learn how to play a best response against a fixed opponent, as was applied to One-Card Poker: a small yet strategic game. Experiments show that two adaptive players playing against each other converge to a Nash equilibrium. However, when using LH to solve extensive form games, the hypothesis space is the set of realization plans (see section 2.1.3) which is difficult to define a potential function for; in this case, an alternate optimization form of the algorithm can be used instead. In the optimization form, $F(s)$ and its gradient are computed by solving an optimization problem. However, this optimization problem takes the form of a quadratic program, and one must be solved at each step.

2.3 Monte Carlo Sampling and Game-Playing

The Monte Carlo technique is a general term used for methods or algorithms that approximate values through sampling. For example, a Monte Carlo algorithm for the bandit problem might try each arm a certain number of times, n, receiving x_i on each trial, and estimate its mean return to be $\frac{\sum_i^n x_i}{n}$. When n is large enough, the law of large numbers

assures that this sample mean is a good estimate for the true mean of the underlying distribution. Given the estimated means of each arm the algorithm would then reason about them; for example, after a certain number of exploration trials, by choosing the one with the highest estimated mean from then on. The quantity sought may be computed exactly, but only with significant resources; Monte Carlo methods are often used when the computation is impractical and an approximation suffices since it is usually less costly [26; 4]. The main idea can be extended to many settings; in extensive games or game trees, either actions or histories are sampled and the sample mean payoff gives an estimate for the expected payoff of a state or history.

As an example, consider a game state at the top of game tree (start of the game); suppose we want to know the expected payoff received when playing action a and then both players follow σ_1 and σ_2. This value can be computed precisely by traversing the entire sub-tree under a. The expected payoff can also be estimated by running simulations until the end of the game, sampling an action from σ_1 or σ_2 at each game state, and computing the mean payoff.

2.3.1 Importance Sampling

Importance sampling is a technique used to measure the expected value of a quantity that is made up of random variables drawn from a distribution that is either impossible or otherwise inconvenient to sample from [116]. To do so, a different sampling distribution is used and then a correction is made for the change in distribution. For a (discrete) random variable $X \sim f$ that takes on values in a finite set $\mathcal{X}$, suppose we want to measure:

$$Q = \mathbb{E}[h(X)] = \sum_{x \in \mathcal{X}} h(x) f(x)$$

A Monte Carlo estimate would take a sample $(x_1, x_2, \cdots, x_n)$ from $X \sim f$ and use the values as estimates for the sum to compute an overall estimate

$$\tilde{Q}_{MC} = \frac{1}{n} \sum_{i=1}^{n} h(x_i)$$

However, f can be difficult or impossible to sample from. Suppose g is a probability density function that is easy to sample from. We can redefine our quantity

$$Q = \sum_{x \in \mathcal{X}} h(x) f(x) = \sum_{x \in \mathcal{X}} h(x) f(x) \frac{g(x)}{g(x)} = \sum_{x \in \mathcal{X}} \frac{h(x) f(x)}{g(x)} g(x) = \mathbb{E}_g(Y),$$

where $Y = h(X)f(X)/g(X)$ and $\mathbb{E}_g(Y)$ is expected value of Y under probability distribution g. Now, we can instead sample $X_1, \cdots, X_n \sim g$ and estimate Q by

$$\tilde{Q}_{IS} = \frac{1}{N}\sum_{i=1}^{N} Y_i = \frac{1}{N}\sum_{i=1}^{N} \frac{h(X_i)f(X_i)}{g(X_i)}.$$

Here, the value of $f(X_i)/g(X_i)$ boosts the *importance* of $h(X_i)$ with respect to estimating the desired quantity. The aim, if possible, is to choose a distribution g such that the samples from g will be biased towards the most important ranges of X_i for estimating the desired quantity.

2.3.2 Perfect Information Games, Game-Playing, and Search

Game-playing is different than game-solving in that it seeks to answer the research question "How can one design an agent that plays game x well?" Computing a Nash equilibrium before employing its strategies during play is one approach to designing computer a Poker-playing agent. However, there are other approaches and algorithms that are used for decision-making in game-playing. The decision of which approach to take usually depends the size of the game, whether it has perfect or imperfect information, and the competition structure (*e.g.,* whether offline computation in advance is allowed.)

In this subsection, we define perfect information games and search. It is important to note that the literature and corresponding community use different terms to refer to the same concepts presented earlier in this chapter. To remain consistent with this literature, we use the appropriate notation for the context. In particular, we will use the notation in this subsection for chapters 6 and 7. To avoid confusion, we will emphasize the similarities in these concepts and link both names explicitly.

A perfect information game is a special type of extensive game: specifically, a perfect information game is one without any hidden information known only by a strict subset of the players. Chess, Checkers, and Go are classical examples of perfect information games without stochasticity (without chance nodes) whereas Backgammon is a classical example of a perfect information game with stochasticity. In perfect information games, information sets each contain a single history each are instead referred to as **states**. At each state $s \in \mathcal{S}$, there is an optimal action $a \in A(s)$ that is part of a max-min or min-max (equilibrium) strategy[7]. In imperfect information games, we described strategies $\sigma_i(I)$ as distributions (mixes) over the actions available $A(I)$. In perfect information games, there is no need to

[7] $\mathcal{S}$ is the analogue to $\mathcal{I}$.

mix over your actions to mislead the opponent. Therefore, most algorithms search for a single optimal action rather than entire strategies.

A finite, two-player zero-sum game of perfect information can be described as a tuple $(\mathcal{S}, Z, \mathcal{A}, \mathcal{P}, u_1, s_1)$, which we now define as a specific kind of Markov Decision Process (MDP). The state space $\mathcal{S}$ is a finite, non-empty set of states, with $Z \subseteq \mathcal{S}$ denoting the set of terminal states. As before, the action space $\mathcal{A}$ is a finite, non-empty set of actions. The transition probability function $\mathcal{P}$ assigns to each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ a probability measure over $\mathcal{S}$ that we denote by $\mathcal{P}(\cdot \mid s, a)$. The utility function $u_1 : Z \mapsto [v_{\min}, v_{\max}] \subset \mathbb{R}$ gives the utility of player 1, with v_{min} and $v_{\max}$ denoting the minimum and maximum possible utility respectively. Since the game is zero-sum, the utility of player 2 in any state $s \in Z$ is given by $u_2(s) = -u_1(s)$. The player index function $P : \mathcal{S} \to \{1, 2\}$ returns the player to act in given state s if $s \in \mathcal{S} \setminus Z$, otherwise it returns 1 in the case where $s \in Z$ to emphasize that the payoffs at the leaves are with respect to player 1.

Each game starts in the initial state s_1 with $P(s_1) = 1$ and proceeds as follows; for each time step $t \in \mathbb{N}$: first, player $P(s_t)$ selects an action $a_t \in \mathcal{A}$ in state s_t, with the next state s_{t+1} generated according to $\mathcal{P}(\cdot \mid s_t, a_t)$. Player $P(s_{t+1})$ then chooses a next action and the cycle continues until some terminal state $s_T \in Z$ is reached. At this point player 1 and player 2 receive a utility of $u_1(s_T)$ and $u_2(s_T)$ respectively.

The **minimax value** of a state $s \in \mathcal{S}$ is defined by

$$V(s) = \begin{cases} \max\limits_{a \in \mathcal{A}} \sum\limits_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a)\, V(s') & \text{if } s \notin Z \text{ and } P(s) = 1 \\ \min\limits_{a \in \mathcal{A}} \sum\limits_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a)\, V(s') & \text{if } s \notin Z \text{ and } P(s) = 2 \\ u_1(s) & \text{otherwise,} \end{cases} \tag{2.19}$$

Note that we always treat player 1 as the player maximizing $u_1(s)$ (*Max*), and player 2 as the player minimizing $u_1(s)$ (*Min*). In games without stochasticity (no chance nodes) $\mathcal{P}(s' \mid s, a)$ is simply considered to be 1 if the next state following s is s' and 0 otherwise. Since we apply our techniques to games with stochasticity in Chapters 6 and 7, we present the general case here.

In most large games, computing the minimax value for a given game state is intractable. Because of this, an often used approximation is to instead compute the *finite horizon minimax value*. This requires limiting the recursion to some fixed depth $d \in \mathbb{N}$ and applying a heuristic evaluation function $h_i : \mathcal{S} \mapsto [v_{\min}, v_{\max}]$ when this depth limit is reached. Thus given a heuristic evaluation function $h_1(s) : \mathcal{S} \to \mathbb{R}$ defined with respect to player 1 that

satisfies the requirement $h_1(s) = u_1(s)$ when $s \in Z$, the finite horizon minimax value is defined recursively by

$$V_d(s) = \begin{cases} \max_{a \in \mathcal{A}} V_d(s,a) & \text{if} \quad d > 0, s \notin Z, \text{ and } P(s) = 1 \\ \min_{a \in \mathcal{A}} V_d(s,a) & \text{if} \quad d > 0, s \notin Z, \text{ and } P(s) = 2 \\ h_1(s) & \text{otherwise,} \end{cases} \tag{2.20}$$

where

$$V_d(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a)\, V_{d-1}(s'), \tag{2.21}$$

For sufficiently large d, $V_d(s)$ coincides with $V(s)$. The quality of the approximation depends on the heuristic evaluation function, the search depth parameter d, and the domain.

The minimax search algorithm performs a shallow search of the game tree online reporting the best possible move given a particular search depth and estimated payoff [93]. An example minimax tree for a perfect information game without chance nodes is shown in Figure 2.4. The current player's state is represented by the root of this tree; they are trying to maximize their payoff and their opponent is trying to minimize it. The search algorithm looks ahead by using depth-limited depth-first search, recursively alternating minimizing and maximizing the values returned by each move.

Alpha-beta pruning is an optimization that reduces the size of the minimax tree by cutting out subtrees that will not change the value returned. At node (a) in Figure 2.4, the parent has already determined an action that gives a utility of 5. Therefore, once the value of 8 is found at node (a) the value returned could only get larger by searching the other actions; the parent will choose 5 regardless. Similarly, at node (b) since the algorithm is minimizing, the value returned could only be smaller than 4; the entire subtree is pruned. In practice, these cutoff points are implemented by maintaining a lower bound (α) and an upper bound (β) at each node, starting at $-\infty, +\infty$. At a Max node, if a value v returned by a recursive traversal of a subtree below an action and $v > \alpha$, then the new lower bound is set to v. Similarly at Min nodes, the upper bound can be reduced as subtrees are visited. This bound information is propagated to the children recursively. If the $[\alpha, \beta]$ *window* (interval) ever becomes empty due to $\beta < \alpha$ then any future search is provably wasteful as a better value has already been found. In the example above, at node (a) the parent sends down a window of $[-\infty, 5]$, and a value of 8 is returned by the subtree causing a cutoff of every other subtree at that node.

The minimax tree represents a subtree of the full game tree; payoff values at the leaves

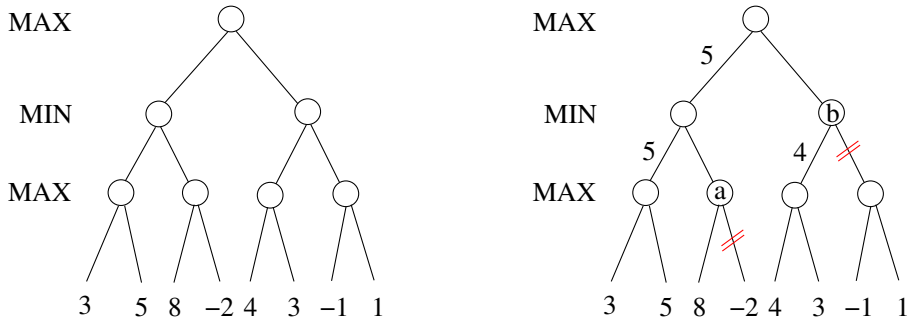


Figure 2.4: An example minimax tree for a perfect information game without chance node. Alpha-beta pruning is demonstrated on an alternating move game.

are estimates returned by the heuristic evaluation function h_1. Consider now, an algorithm we will call **full minimax**, which is not a depth-limited approximation; in other words, rather than stopping early and returning heuristic evaluations, it continues to the end of the game in every case, returning the true payoff values. The path that full minimax recommends, the so-called *principal variation*, would be one taken by players employing one particular pure Nash equilibrium profile. In addition, the path computed by full minimax is a path taken by a (pure behavioral strategy) solution to the sequence-form linear program constructed from Equation 2.8. Even the regret minimization techniques above can be applied to perfect information games, since every finite perfect information extensive game (with perfect recall) is a finite imperfect information extensive game (with perfect recall) with the added restriction that $|I| = 1$ for all $I \in \mathcal{I}_i$. The problem is that the space and time complexities are functions of $\mathcal{I}$ (hence $\mathcal{S}$). In perfect information games, a pure Nash equilibrium always exists; due to the interchangeability of Nash equilibria in two-player zero-sum games, it suffices then from some state s to search for a single action that is part of a pure equilibrium strategy. Minimax and other search algorithms take advantage of a specific property of perfect information games to make better use of computation time.

Other search-based techniques are also used in the perfect information setting for solving games, *e.g.,* pattern databases and proof number search [25; 3; 98].

Monte Carlo Sampling in Game Tree Search

Monte Carlo methods are also commonly used in search. In single-agent problems, a strategy is commonly referred to as a **policy**, denoted π. In general, π can be a stochastic policy and there may be stochasticity in the transitions from state to state (just like a mixed strategy), so S_t is a random variable representing the state the agent is in at time t. At each time t,

the agent chooses an action $A_t \sim \pi(\cdot \mid S_t)$. In the general case, when problems are modeled as MDPs, the system then responds with a state-reward pair $(S_{t+1}, R_{t+1}) \sim \mathcal{P}(\cdot \mid S_t, A_t)$, where $S_{t+1} \in \mathcal{S}$ and $R_{t+1} \in [v_{\min}, v_{\max}]$.

A canonical example of online Monte-Carlo planning is 1-ply rollout-based planning [9]. It combines a *default policy* π with a one-ply look-ahead search. At each time $t < n$, given a starting state s_t, for each $a_t \in \mathcal{A}$ and with $t < i < n$, $\mathbb{E}\left[X_{s_t,a_t} \mid A_i \sim \pi(\cdot \mid S_i)\right]$ is estimated by generating trajectories S_{t+1}, R_{t+1}, $\ldots$, A_{n-1}, S_n, R_n of agent-system interaction. From these trajectories, sample means $\bar{X}_{s_t,a_t}$ are computed for all $a_t \in \mathcal{A}$. The agent then selects the action $A_t = \arg\max_{a_t \in \mathcal{A}} \bar{X}_{s_t,a}$, and observes the system response (S_{t+1}, R_{t+1}). This process is then repeated until time n. Under some mild assumptions, this technique is provably superior to executing the default policy on its own [9].

One of the main advantages of rollout based planning compared with exhaustive depth-limited search is that a much larger search horizon can be used. This is particularly useful in episodic tasks where $n \approx 1000$ or less. Let π^* be the optimal policy which maximizes the expected return[8]. The disadvantage of rollout-based planning is that if π is suboptimal, then $\mathbb{E}\left[X_{s_t,a} \mid A_i \sim \pi(\cdot \mid S_i)\right] < \mathbb{E}\left[X_{s_t,a} \mid A_i \sim \pi^*(\cdot \mid S_i)\right]$ for at least one state-action pair $(s_t, a) \in \mathcal{S} \times \mathcal{A}$, which implies that at least some value estimates constructed by 1-ply rollout-based planning are biased. This can lead to mistakes which cannot be corrected through additional sampling. The bias can be reduced by incorporating more knowledge into the default policy, however this can be both difficult and time consuming.

Monte Carlo Tree Search (MCTS) is a recent technique used to sample actions on-line [18; 14]. MCTS is a general technique for Monte Carlo game-playing inspired by the Upper Confidence Bounds for Trees (UCT)[9] algorithm [61]. MCTS is a best-first search algorithm that incrementally builds a tree in memory. MCTS constructs asymptotically consistent estimates of the return under the optimal policy from simulation trajectories. It uses a default policy to generate trajectories of agent-system interaction. The construction of a search tree is also interleaved within this process. Initially, the search tree consists of a single node, which represents the current state s_t at time t. One or more *simulations* are then performed. We will use $\mathcal{T}_m \subset \mathcal{S}$ to denote the set of states contained within the search tree after $m \in \mathbb{N}$ simulations. Associated with each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ is an estimate $\bar{X}^m_{s,a}$ of the return under the optimal policy and a count $T^m_{s,a} \in \mathbb{N}$ representing the

[8] In a 2-player adversarial setting, π^* is a minimax-optimal strategy, as described in Section 2.3.2.

[9] In this thesis, our techniques are applicable to the general setting and so we choose to refer to MCTS rather than to UCT. However, all of our techniques are applicable to UCT and much of the algorithm described in this section was initially presented in the original paper by Kocsis & Szepesvári (2006). As such, we may use the terms MCTS and UCT interchangeably.

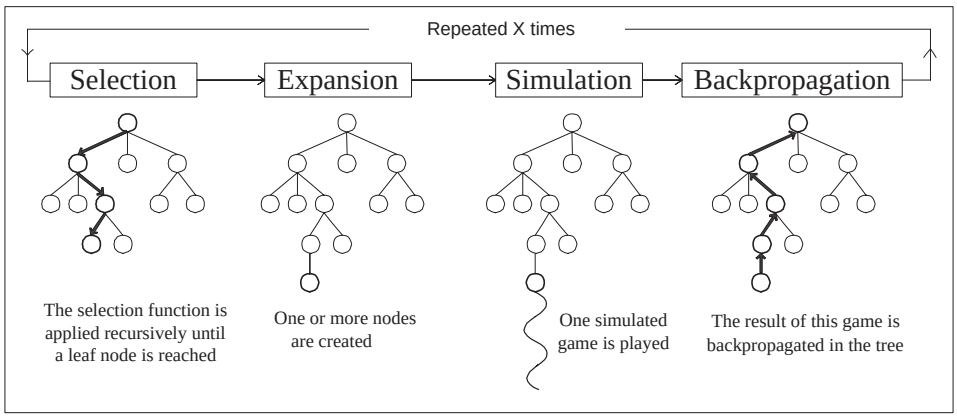


Figure 2.5: A summary of the Monte Carlo Tree Search (MCTS) algorithm. Image provided by Guillaume Chaslot.

number of times this state-action pair has been visited after m simulations, with $T^0_{s,a} = 0$ and $\bar{X}^0_{s,a} = 0$.

Each simulation can be broken down into four phases, *selection*, *expansion*, *rollout* and *backup*. Selection involves traversing a path from the root node to a leaf node in the following manner: for each non-leaf, internal node representing some state s on this path, the UCB [6] criterion is applied to select an action until a leaf node corresponding to state s_l is reached. If $\mathcal{U}(\mathcal{B}_s)$ denotes the uniform distribution over the set of unexplored actions, define $\mathcal{B}^m_s = \{a \in \mathcal{A} : T^m_{s,a} = 0\}$, and $T^m_s = \sum_{a\in\mathcal{A}} T^m_{s,a}$, then UCB at state s selects

$$A^{m+1}_s = \arg\max_{a\in\mathcal{A}} \bar{X}^m_{s,a} + C\sqrt{\log(T^m_s)/T^m_{s,a}}, \tag{2.22}$$

if $|\mathcal{B}^m_s| = \emptyset$, or $A^{m+1}_s \sim \mathcal{U}(\mathcal{B}^m_s)$ otherwise. The ratio of exploration to exploitation is controlled by the positive constant $C \in \mathbb{R}$. In the case of more than one maximizing action, ties are broken uniformly at random. Provided s_l is non-terminal, the expansion phase is then executed, by selecting an action $A_l \sim \pi(\cdot \mid s_l)$, observing a successor state $S_{l+1} = s_{l+1}$, and then adding a node to the search tree so that $\mathcal{T}_{m+1} = \mathcal{T}_m \cup \{s_{l+1}\}$. Higher values of c increase the level of exploration, which in turn leads to more shallow and symmetric tree growth. The rollout phase is then invoked, which for $l < i < n$, executes actions $A_i \sim \pi(\cdot \mid S_i)$. At this point, a complete agent-system execution trajectory $(a_t, s_{t+1}, r_{t+1}, \ldots, a_{n-1}, s_n, r_n)$ from s_t has been realized. The backup phase then assigns, for $t \le k < n$,

$$\bar{X}^{m+1}_{s_k,a_k} \leftarrow \bar{X}^m_{s_k,a_k} + \tfrac{1}{T^m_{s_k,a_k}+1}\left(\sum_{i=k+1}^{n} r_i - \bar{X}^m_{s_k,a_k}\right), \quad T^{m+1}_{s_k,a_k} \leftarrow T^m_{s_k,a_k} + 1,$$

to each $(s_k, a_k) \in \mathcal{T}_{m+1}$ occurring on the realized trajectory. Notice that for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, the value estimate $\bar{X}^m_{s,a}$ corresponds to the average return of the realized simulation trajectories passing through state-action pair (s, a). After the desired number of simulations m has been performed in state s_t, the action with the highest expected return $a_t = \arg\max_{a \in \mathcal{A}} \bar{X}^m_{s_t,a}$ is selected. With an appropriate value of C, as $m \to \infty$, the value estimates converge to the expected return under the optimal policy [61]. However, due to the stochastic nature of the algorithm, each value estimate $\bar{X}^m_{s,a}$ is subject to error, in terms of both bias and variance, for finite m.

MCTS has been successfully applied to game-playing and have received much attention. One notable example is the game of Go where it has allowed creation of strong computer players [31; 30]. In the general game-playing competition [32], MCTS-based players have been shown to have strong performance [27]. In this setting no model of the game is known apriori, so sampling online can be used to estimate the quality of each move at a tree node.

What about imperfect information games? The search and game-playing communities have been applying search methods (including Monte Carlo Tree Search methods) to imperfect information games. The main technique used here is called **determinization**: when faced with a decision, sample a state from the current information set, applying a search method to this sampled perfect information game rooted at the sampled state, and aggregating the results. Determinization-based techniques have been applied in with empirical success in Bridge [37], Skat [101; 72], Dou Di Zou [89], Magic: The Gathering [24], and Scotland Yard [80]. Perfect Information Monte Carlo (PIMC) search uses determinization and minimax-based classical search, and was applied to Skat and Poker [73]. A recent algorithm, Information Set Monte Carlo Tree Search, searches the players' individual information set trees rather than the perfect information game tree [23].

Unfortunately, determinization-based MCTS players playing against each other in an imperfect information game may not converge to a Nash equilibrium when obtaining a mixed strategy by normalizing visit counts. One counter-example is shown on a small game of biased Rock Paper Scissors [103]. Another empirical example is shown on Kuhn Poker [88], but in this case the algorithm seems to avoid dominated strategies. Whether determinization-based MCTS algorithms could be modified so that equilibria are approached over time remains an open question. However, in the perfect information setting, given infinite resources UCT will recommend an action that is part of an equilibrium [61], even in games with more than two players [107].

Chapter 3

Games

In this chapter, we describe the extensive form games that will be referred to throughout the thesis. The games presented here are used as domains for empirical investigations of the algorithms that follow.

3.1 Imperfect Information Games

Recall that an imperfect information game is an extensive game where some information may only be revealed to a strict subset of the players. In this section, we describe the imperfect information games used throughout the thesis.

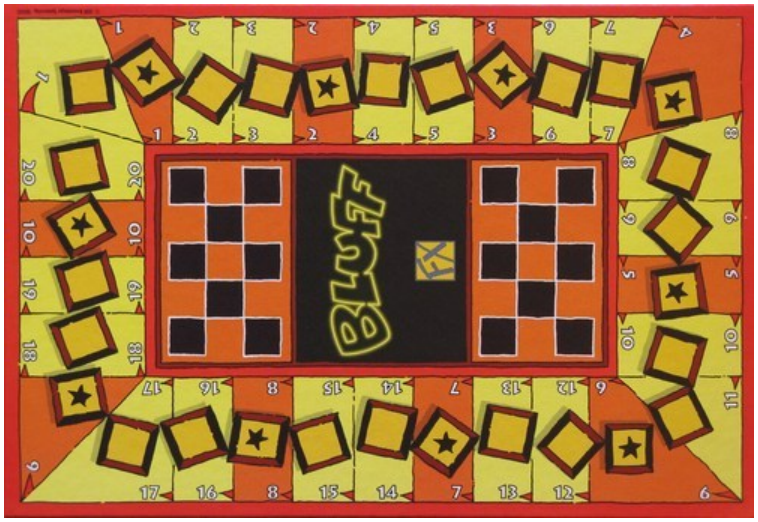


Figure 3.1: Bluff's game board.

3.1.1 Bluff

Bluff, also known as Liar's Dice, Dudo, and Perudo, is a dice-bidding game. In our version, Bluff(D_1,D_2), each die has six sides with faces ⚀, ⚁, ⚂, ⚃, ⚄, and a star: ⋆. Each player i rolls D_i of these dice and looks at them without showing them to their opponent. Each round, players alternate by bidding on the outcome of all dice in play until one player "calls bluff", *i.e.,* claims that their opponent's latest bid does not hold. A bid consists of a quantity of dice and a face value. A face of ⋆ is considered "wild" and counts as matching any other face. For example, the bid 2-⚄ represents the claim that there are at least two dice with a face of ⚄ or ⋆ among both players' dice. To place a new bid, the player must increase either the quantity or face value of the current bid (or both); in addition, lowering the face is allowed if the quantity is increased. The losing player removes a number of dice L from the game and a new round begins, starting with the player who won the previous round. The number of dice removed is equal to the difference in the number of dice claimed and the number of actual matches; in the case where the bid is exact, the calling player loses a single die. When a player has no more dice left, they have lost the game. A utility of $+1$ is given for a win and -1 for a loss.

We refer to Bluff(D_1, D_2) as a single-round of the full game. The full game is composed of multiple rounds, where each subsequent round is considered a subgame. For example, suppose after a game of Bluff(D_1, D_2) the loser loses L dice; then the next round of the is either Bluff(D_1, $D_2 - L$) or Bluff($D_2, D_1 - L$), depending on which player lost. Therefore, in Bluff(2,2) the game values of Bluff(1,1) are precomputed and then used as payoffs at the leaves of Bluff(2,1), whose game values are then re-used as leaf payoffs for Bluff(2,2). Similarly for Bluff games with $D_i \geq 3$. In Bluff(2,2), the total number of histories $|H| \approx 10^{10}$.

Unlike games like limit Texas Hold'em Poker, the number of information sets in Bluff is due to the number of possible bidding sequences. There are $\binom{D_i+5}{5}$ unique outcomes when player i rolls their D_i dice [60]. The number of possible bids is $6(D_1 + D_2)$. Since the bidding sequence does not depend on either player's dice, a bidding sequence can be directly represented as a binary string. For example, a bidding sequence of 1-⚀, 1-⚂, 1-⚃, 1-⋆, 2-⚀ can be represented by the binary string `1011011`. Therefore, there are $2^{D_1+D_2}$ unique bidding sequences. Each bidding sequence belongs to exactly one player and so the number of information sets per bidding sequence is equal to the number of different unique die-roll outcomes. Therefore, the total number of information sets belonging to both

players in a game of Bluff, when $D_1 = D_2$, is $\binom{D_i+5}{5}2^{6D_1+6D_2}$. Bluff(1,1), Bluff(2,2), and Bluff(3,3) have 24576, 352 million, and 3.848×10^{12} information sets, respectively.

3.1.2 One-Card Poker

One-Card Poker, abbreviated OCP(N), is a simplified two-player game of Poker described by Geoffrey Gordon [40; 39]. Each player is dealt a single card from a deck of N cards. The first player can bet or pass. If the first player bets, the second player can call or fold. If the first player passes, the second gets the option to bet or pass, where a bet causes the first player to have to decide whether to call or fold. The game ends on two passes, a call, or a fold. On a fold, the betting player gets 1$ (and the folding player loses $1). On two passes or a call, the player with the highest card wins $2 or $1 respectively (and the other player loses the same amount). In OCP(500), $|H| \approx 22.4 \cdot 10^6$ and there are approximately 2000 information sets among both players. Note that OCP(3) is equivalent to Kuhn's simplified version of poker [65].

3.1.3 Die-Roll Poker

Die-roll poker, abbreviated DRP, is a simplified two-player poker game that uses dice rather than cards. To begin, each player antes one chip to the pot. There are two betting rounds, where at the beginning of each round, players roll a private six-sided die. The game has imperfect information due to the players not seeing the result of the opponent's die rolls. During a betting round, a player may fold (forfeit the game), call (match the current bet), raise (increase the current bet) by a fixed number of chips, with a maximum of two raises per round. In the first round, raises are worth two chips, whereas in the second round, raises are worth four chips. If both players have not folded by the end of the second round, a showdown occurs where the player with the largest sum of their two dice wins all of the chips in the pot.

3.1.4 Imperfect Information Goofspiel

Goofspiel(N) is a two-player card game where each player is given a private hand of bid cards with values 1 to N. A different deck of N point cards is shuffled and the cards are placed face up in a stack. On their turn, each player bids for the top point card by choosing a single card in their hand and revealing the chosen card simultaneously. Whoever bid the highest value gets the point card and adds the point total to their score. In the case of a tie, the point card is discarded. Whether they won or lost the bid, both players discard the card

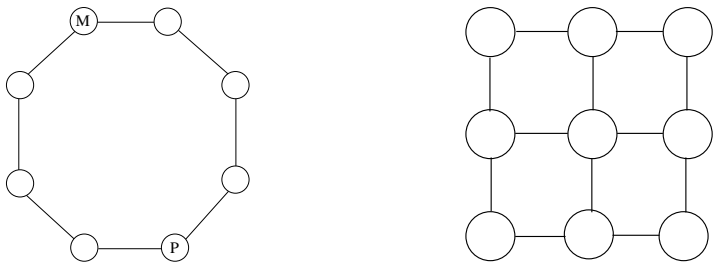


Figure 3.2: An 8-node circle graph and four-connected 3-by-3 graph used in Princess and Monster games.

they used to bid. This bidding process is repeated until players have no bid cards left. The player with the highest score wins and receives a utility of 1. The losing player receives a utility of -1.

Our imperfect information version of Goofspiel has two changes. First, only whether the player won or lost a bid is revealed, not the cards that were used to bid. Second, the point card stack is sorted in a known (descending) order rather than being randomly shuffled. The former change increases the degree of imperfect information in the game, since the original Goofspiel is a simultaneous move game only having very short-term imperfect information. The latter change allows us to better control the size of the game.

In Goofspiel(7), $|H| \approx 98.3 \cdot 10^6$ and $|\mathcal{I}| \approx 3.3 \cdot 10^9$.

3.1.5 Princess and Monster

Princess and Monster, abbreviated PAM($t_{\max}, G$), is a pursuit-evasion game based on a similar continuous game [49, Research Problem 12.4.1]. The monster and princess take turns moving to nodes on a graph G in a "dark room" where neither knows the location of the other. Unlike the original problem, both princess and monster are restricted to moving to adjacent nodes. A time step counter is incremented after the monster moves and then the princess moves, where $t = 0$ initially. There is a time limit; if the monster catches the princess (both occupy the same node in the graph) within the time limit it gets a payoff of $t_{\max} - t + 1$. Otherwise, its payoff is $-t_{\max}$. The princess gets the inverse payoff of the monster: $+t_{\max}$ if uncaptured or a negative value based on the number of time steps she remained uncaptured $-t_{\max} + t - 1$.

We consider two different graphs shown in Figure 3.2: the 8-node circle graph with opposing starting positions and a 9-node 3-by-3 grid world graph with random starting positions.

In Princess and Monster on the 3-by-3 grid, $|H| \approx 91.4 \cdot 10^6$ and $|\mathcal{I}| \approx 20 \cdot 10^3$.

3.1.6 Latent Tic-Tac-Toe

Latent Tic-Tac-Toe is a more interesting version of the classic two-player game where players take turns claiming squares in a 3x3 grid. In Latent Tic-Tac-Toe each players' moves are "delayed". Whenever it is a player's turn they write down their move secretly and it is only revealed to take effect at the beginning of their next turn. This means the opponent will have to take make their own move without knowing the opponent's latest. The delayed moves are valid if the target square is empty at the time it is revealed. If a delayed move is not valid when it is revealed then the move is lost. The goal of each player remains to get three squares in a straight line (horizontal, vertical, or diagonal). The winning player receives a utility of +1 and the losing player receives -1. In Latent-Tic-Tac-Toe, $|H| \approx 70.4 \cdot 10^6$ and $|\mathcal{I}| \approx 16 \cdot 10^6$.

3.1.7 Phantom Tic-Tac-Toe

As in regular tic-tac-toe, phantom tic-tac-toe (PTTT) is played on a 3-by-3 board, initially empty, where the goal is to claim three squares along the same row, column, or diagonal. However, in PTTT, players' actions are private. Each turn, a player attempts to take a square of their choice. If they fail due to the opponent having taken that square on a previous turn, the same player keeps trying to take an alternative square until they succeed. Players are not informed about how many attempts the opponent made before succeeding. The game ends immediately if there is ever a connecting line of squares belonging to the same player. The winner receives a payoff of $+1$, while the losing player receives -1. In PTTT, the total number of histories $|H| \approx 10^{10}$ and $|\mathcal{I}| \approx 5.6 \cdot 10^6$.

3.2 Perfect Information Games

Recall that a perfect information game is an extensive game where $|I| = 1$ for all $I \in \mathcal{I}_i$ for every player i. In other words, information is never known only to a strict subset of the players.

3.2.1 Pig

Pig is a turn-based jeopardy dice game that can be played with one or more players [97]. Players roll two dice each turn and keep a turn total. At each decision point, they have two actions, roll and stop. If they decide to stop, they add their turn total to their total score.

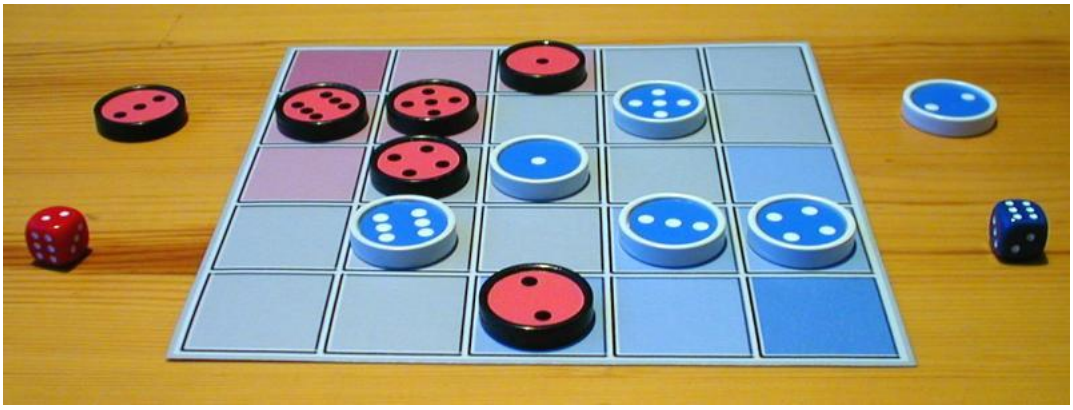


Figure 3.3: EinStein Würfelt Nicht! player board and components. Source: `http://boardgamegeek.com/image/128344/einstein-wurfelt-nicht`.

Normally, dice rolls add to the players turn total, with the following exceptions: if a single ⚀ is rolled the turn total will be reset and the turn ended; if a ⚀⚀ is rolled then the player's turn will end along with their *total* score being reset to 0. These possibilities make the game highly stochastic.

3.2.2 EinStein Würfelt Nicht!

EinStein Würfelt Nicht! (EWN) [1] is a tactical dice game played on a 5 by 5 grid. Each player, red or blue, starts with their 6 dice or pieces (labeled ⚀, ⚁, ..., ⚅) in the top left corner squares or bottom-right corner squares. The goal is to reach the opponent's corner square with a single die or capture every opponent piece. Each turn starts with the player rolling a white die off the board which indicates which of their dice they can move this turn. Pieces can only move toward the opponent's corner square or off the board; if they move a die over a square containing another die (belonging to them or the opponent), it is captured. EWN is a popular game played by humans and computer opponents on the Little Golem online board game site `http://www.littlegolem.net`.

3.2.3 Can't Stop

Can't Stop is a dice game where the goal is to obtain three complete columns by reaching the highest level in each of the 2-12 columns [95]. On a player's turn, they choose to either *roll* or *stop*. If the player chooses to roll, they roll 4 dice and from the four dice they group the dice into two pairs of two dice each. For example, suppose the player rolls ⚀⚀⚂⚄, the valid pairings are (⚀⚀, ⚂⚄), and (⚀⚂, ⚀⚄), or simply $(2, 7)$ and $(4, 5)$; the player

[1] Translated as "Einstein does not play dice!"

Figure 3.4: Can't Stop player board and components.

chooses one of these and moves their *turn progress markers* (white tokens in Figure 3.4) up one level in each column. A player has only three turn progress markers that they can use each turn; when all three are on the board, no more new ones can be placed that turn and the white markers may not be moved to different columns that turn. The player then chooses to roll or stop again. If the dice are rolled and no legal pairing (*i.e.,* a pair that sums to two columns, at least one of which the player may advance on) can be made, the player loses all of the progress made towards completing columns on this turn. If the player chooses to stop, their turn ends and they move their *permanent player markers* in their color up to the level reached by the turn progress markers and then remove the turn progress markers from the board. The opponent takes their turn either after the player stops or after the player rolls no legal pairings. A key component of the game involves correctly assessing the risk associated with not being able to make a legal dice pairing given the current board configuration.

3.2.4 Dominion

Dominion is a popular turn-based, deck-building card game [108]. Each player incrementally builds a deck of cards, which initially contains seven 1-money cards (Coppers) and three 1-victory point cards (Estates).

On their turn, a player draws a hand of 5 cards. The player is allowed to play a single action card (silver cards, *e.g.,* Smithy from Figure 3.5), then can buy a single card from the common pool of money cards, victory point cards, and 10 unique action card stacks. Action cards have certain effects that allow you to buy more cards, get more money, draw more cards, and earn victory points. Bought cards are placed into the discard pile, which is reshuffled into the deck when it runs out. The victory point cards have no effect during

Figure 3.5: Dominion game cards.

the game. At the end of each turn, players take any remaining cards in their hand and place them into their discard pile. Players alternate turns until the stack of Province cards runs out or there are three of ten stacks of action cards run out. The goal is to get more victory points than your opponents.

Due to the large variance in expected utility between each 5-card combination that can be drawn, stochasticity has a large effect on the outcome of a single game of Dominion. The amount of money cards combined determines what card the player can add to their deck; some unlucky draws early in the game could mean not being able to buy the better money cards and leading to the strength of the deck growing slower than the opponent's. In addition, if a player draws no action cards or more action cards than one can play in a turn, they can lose potential action effects which are often significant.

Chapter 4

Monte Carlo Counterfactual Regret Minimization

In this chapter, we present Monte Carlo sampling for counterfactual regret minimization. The contents of this chapter are based on the paper [70], which was produced in collaboration with Martin Zinkevich, Kevin Waugh, and Michael Bowling. We will most often refer to the technical report version of the paper [71], which contains additional material referred to by this chapter.

Recall from Section 2.2.2, the counterfactual regret minimization (CFR) algorithm (Algorithm 1) is an iterative procedure where at time step $t \in \{1, 2, \ldots, T\}$, players employ strategy profile (σ_1^t, σ_2^t). The algorithm computes a regret $r(I, a)$ of *not* taking action $a \in A(I)$ for every action a and every information set $I \in \mathcal{I}_i$. A regret minimization process then computes a new profile $(\sigma_1^{t+1}, \sigma_2^{t+1})$, t is incremented, and the procedure continues. One critical aspect of the CFR algorithm to remember, for the purposes of equilibrium approximation, is that the strategy approaching equilibrium is the average strategy $\bar{\sigma}^T$; the strategies σ^t are discarded after each trial. The algorithm maintains tabulated cumulative regrets for each action at each information set, which determines the subsequent strategies.

4.1 Sampled Counterfactual Regret

Recall that the definition of counterfactual value for player i using profile σ (Equation 2.15) is:

$$v_i(\sigma, I) = \sum_{z \in Z_I} \pi^{\sigma}_{-i}(z[I]) \pi^{\sigma}(z[I], z) u_i(z).$$

This value represents an expected utility to player i when reaching information set I weighted by the probability of the opponent playing such that I is reached. Recall that the counterfactual regret of not playing with a strategy $\sigma^t_{I \to a}$ that chooses a particular action a at I but

follows σ_i^t everywhere else (Equation 2.17) is:

$$r^t(I, a) = v_i(\sigma^t_{I\to a}, I) - v_i(\sigma^t, I). \tag{4.1}$$

The main idea behind Monte Carlo CFR (MCCFR) is to instead define unbiased estimators of counterfactual value and regret. Each iteration, MCCFR then samples only parts of the game tree and applies the regret minimization process to the sampled subtree using the estimated values instead of the true values. Since the estimates are unbiased, the estimated values approach the true values in expectation. In this chapter, we formalize these notions. We show that as long as care is taken in how to sample these subtrees, approaching approximate equilibria is possible with high probability. We prove regret bounds (that directly imply convergence rates) for two particular sampling schemes, show empirical results, and survey a number of applications of MCCFR in the current literature.

Define $\mathcal{Q} = \{Q_1, Q_2, Q_3, \ldots\}$ be a set of subsets of the Z such that $\bigcup_{Q\in\mathcal{Q}} Q = Z$. Note, in particular, that $\mathcal{Q}$ is not necessarily a partition because we do not require $Q_i \cap Q_j = \emptyset$. We refer to an element of $Q_j \in \mathcal{Q}$ as a **block** of terminal histories $Q \subseteq Z$, and the set $\mathcal{Q}$ as the set of all possible blocks. On each iteration t, MCCFR first samples a block Q_j with probability $q_j > 0$ using some sampling scheme. Then, MCCFR computes counterfactual values and regrets for each information set I such that $z \in Q_j, h \sqsubset z, h \in I$. Let

$$q(z) = \sum_{j:z\in Q_j} q_j \tag{4.2}$$

be the probability that z is contained in the block that is sampled at some iteration[1]. The **sampled counterfactual value** is defined as:

$$\tilde{v}_i(\sigma, I|j) = \sum_{z\in Q_j\cap Z_I} \frac{1}{q(z)} \pi^\sigma_{-i}(z[I])\pi^\sigma(z[I], z)u_i(z). \tag{4.3}$$

Note that as long as we guarantee $q(z) \geq \delta > 0$ then the sampled counterfactual regret is well-defined; δ is the lowest probability of sampling any z and will become a critical part of the bounds in Section 4.5.

When the intersection $Q_j \cap Z_I$ is empty, the block contains none of the terminal histories passing through I and hence the value is 0. The sampled counterfactual value $\tilde{v}_i(\sigma, I|j)$ is essentially a function of a random variable whose distribution is defined by the sampling scheme over the possible blocks $\mathcal{Q}$. The following lemma relates the expected value of $\tilde{v}_i(\sigma, I|j)$ with $v_i(\sigma, I)$.

[1] In general, a different sampling scheme may be used at every iteration; here, we intentionally omit t from the definition of q_j as it simplifies the notation. For any statements about $q(z)$ and q_j, assume an implicit $\forall t \in \{1, 2, \ldots\}$.

Lemma 1. $E\left[\tilde{v}_i(\sigma, I|j)\right] = v_i(\sigma, I)$.

Proof.

$$E\left[\tilde{v}_i(\sigma, I|j)\right] = \sum_j q_j \tilde{v}_i(\sigma, I|j) \tag{4.4}$$

$$= \sum_j \sum_{z \in Q_j \cap Z_I} \frac{q_j}{q(z)} \pi^\sigma_{-i}(z[I]) \pi^\sigma(z[I], z) u_i(z) \tag{4.5}$$

$$= \sum_{z \in Z_I} \frac{\sum_{j: z \in Q_j} q_j}{q(z)} \pi^\sigma_{-i}(z[I]) \pi^\sigma(z[I], z) u_i(z) \tag{4.6}$$

$$= \sum_{z \in Z_I} \pi^\sigma_{-i}(z[I]) \pi^\sigma(z[I], z) u_i(z) = v_i(\sigma, I) \tag{4.7}$$

Equation 4.6 follows from the fact that $\mathcal{Q}$ spans Z. Equation 4.7 follows from the definition of $q(z)$ from Equation 4.2. □

Lemma 1 shows that $\tilde{v}_i(\sigma, I|j)$ is an unbiased estimator of $v_i(\sigma, I)$. Therefore, the sampled counterfactual regret values

$$\tilde{r}^t(I, a) = \tilde{v}_i(\sigma^t_{I \to a}, I) - \tilde{v}_i(\sigma^t, I) \tag{4.8}$$

match the true regret values $r^t(I, a)$ in expectation.

A high-level description of the MCCFR algorithm is given in Algorithm 2. Lower-level pseudo-code is given below.

How the blocks Q_j are sampled has a large impact on the theoretical guarantees and empirical performance of the algorithm. These particular definitions of a block set and sampled counterfactual value allows us to generalize the previous algorithms. When $\mathcal{Q} = \{Z\}$ then the sampled counterfactual value is equal to the counterfactual value and the algorithm simplifies to "vanilla" CFR without any sampling at all. If the terminal histories in the same block are grouped together because of realizing the same outcomes at chance nodes, the algorithm simplifies to the chance sampling variant used in previous Poker agents. Additionally, we present two new sampling schemes below.

The time complexity of MCCFR will depend on the sampling scheme used. For instance, in Vanilla CFR the time per iteration is linear in the size of the game tree. In a game that contains a single chance node at the root that has 4 outcomes with equally-sized trees below each outcome, then one iteration of chance-sampled CFR will take roughly a quarter of the time taken by a Vanilla CFR iteration. We will mention the time complexity of each MCCFR algorithm in the corresponding section. On the other hand, every instance of MCCFR (including Vanilla CFR) stores the strategy space, and so the space complexity

Algorithm 2 Monte Carlo Counterfactual Regret Minimization

Require: a sampling scheme $\mathcal{S}$
Initialize regret tables: $\forall I \in \mathcal{I}, \forall a \in A(I) : r_I[a] \leftarrow 0$
Initialize cumulative strategy tables: $\forall I \in \mathcal{I}, \forall a \in A(I) : s_I[a] \leftarrow 0$
Initialize initial profile: $\forall I \in \mathcal{I}, \forall a \in A(I) : \sigma(I, a) \leftarrow 1/|A(I)|$
  for $t \in \{1, 2, 3, \ldots\}$ **do**
    for $i \in N$ **do**
      Sample a block of terminal histories $Q \in \mathcal{Q}$ using $\mathcal{S}$
      for each prefix $z[I]$ of each terminal history $z \in Q$ with $P(z[I]) = i$ **do**
        $\sigma_i(I) \leftarrow$ RegretMatching(r_I) using Eq. 2.18
        for $a \in A(I)$ **do**
          Let $\tilde{r} \leftarrow \tilde{r}(I, a)$, the sampled regret for not taking a
          $r_I[a] \leftarrow r_I[a] + \tilde{r}$
          $s_I[a] \leftarrow s_I[a] +$ AverageStrategyIncrement(s_I, t, σ_i, I, a)

is $O(|\mathcal{C}_1| + |\mathcal{C}_2|)$. When space is a limiting factor, abstraction techniques can be used to decrease the memory consumption; we will discuss abstraction more in Chapter 5.

4.2 Outcome Sampling

In **outcome sampling** MCCFR we choose $\mathcal{Q}$ so that each block contains a single terminal history, *i.e.,* $\forall Q \in \mathcal{Q}, |Q| = 1$. On each iteration we sample one terminal history and only update each information set along that history. The sampling probabilities, q_j must specify a distribution over terminal histories. We will specify this distribution using a *sampling profile*, σ', so that $q(z) = \pi^{\sigma'}(z)$. Note that any choice of sampling policy will induce a particular distribution over the block probabilities $q(z)$. As long as $\sigma'_i(I, a) > \epsilon$, then there exists a $\delta > 0$ such that $q(z) \geq \delta$, thus ensuring Equation 4.3 is well-defined.

The algorithm works by sampling z using policy σ', storing $\pi^{\sigma'}(z)$. The single history is then traversed forward to compute each player's probability of playing to reach each prefix of the history, $\pi_i^{\sigma}(h)$ and then backward to compute each player's probability of playing the remaining actions of the history, $\pi_i^{\sigma}(h, z)$. During the backward traversal, the sampled counterfactual regrets at each visited information set are computed (and added to the total regret).

When using outcome sampling, there are two cases. Either $z[I]a$ is a prefix of z (action a was taken at I in our sampled history), or $z[I]a$ is not a prefix and a was not taken in z.

When we have the former,

$$\tilde{r}(I,a) = \tilde{v}_i(\sigma^t_{(I\to a)}, I) - \tilde{v}_i(\sigma^t, I) \tag{4.9}$$

$$= \frac{u_i(z)\pi^\sigma_{-i}(z[I])\pi^\sigma(z[I]a,z)}{\pi^{\sigma'}(z)} - \frac{u_i(z)\pi^\sigma_{-i}(z[I])\pi^\sigma(z[I],z)}{\pi^{\sigma'}(z)} \tag{4.10}$$

$$= \frac{u_i(z)\pi^\sigma_{-i}(z[I])}{\pi^{\sigma'}(z)}\left(\pi^\sigma(z[I]a,z) - \pi^\sigma(z[I],z)\right) \tag{4.11}$$

$$= W \cdot (\pi^\sigma(z[I]a,z) - \pi^\sigma(z[I],z)) \tag{4.12}$$

where

$$W = \frac{u_i(z)\pi^\sigma_{-i}(z[I])}{\pi^{\sigma'}(z)} \tag{4.13}$$

When $z[I]a$ is not a prefix of z, then $\tilde{v}_i(\sigma^t_{(I\to a)}, I) = 0$, so

$$\tilde{r}(I,a) = 0 - \tilde{v}_i(\sigma^t, I) \tag{4.14}$$

$$= -W \cdot \pi^\sigma(z[I],z) \tag{4.15}$$

There is still the question of how to design the sampling profile σ'. It is sensible to sample information sets that are likely to occur given the players' strategies. The most straightforward way to ensure exploration while doing this is to use an ϵ-greedy approach. When traversing the tree and at information set I, with probability ϵ choose an action uniformly at random, otherwise sample it according to the player's current strategy $\sigma^t(I)$. We call this sampling method **epsilon-on-policy** exploration.

Since only a single history is sampled, each iteration takes time linear in the depth of the tree as is unaffected by the game's branching factor. The downside is having to constantly explore, leading to choosing actions that may clearly never be played.

In outcome sampling a single sampled history can be re-used for both players; we call this the **parallel form** of outcome sampling. Here, we present only the simple case here where the updating player i will alternate, (the **alternating form**), however we will show the difference in practice in Section 4.6.

Before presenting more specific pseudo-code, we still need to discuss computing the average strategy $\bar{\sigma}^T$. Therefore, we present outcome sampling in Algorithm 3 contained in Section 4.4.

4.3 External Sampling

In **external sampling** we sample only the actions of the opponent and chance (those choices external to the player). We have a block $Q_\tau \in \mathcal{Q}$ for each pure strategy of the opponent and

chance, *i.e.,*, for each deterministic mapping τ from $I \in \mathcal{I}_c \cup \mathcal{I}_{N\setminus\{i\}}$ to $A(I)$. The block probabilities are assigned based on the distributions f_c and σ_{-i}, so

$$q_\tau = \prod_{I \in \mathcal{I}_c} \sigma_c(I, \tau(I)) \prod_{I \in \mathcal{I}_{N\setminus\{i\}}} \sigma_{-i}(I, \tau(I)).$$

The block Q_τ then contains all terminal histories z consistent with τ, that is if ha is a prefix of z with $h \in I$ for some $I \in \mathcal{I}_{-i}$ then $\tau(I) = a$. The σ_{-i} used to sample the opponent's and chance's (deterministic) strategy is *not* mixed with exploration as is done in outcome sampling, therefore the sampling is entirely on-policy in this case.

In practice, we will not actually sample τ but rather sample the individual actions that make up τ only as needed. The key insight is that these block probabilities result in $q(z) = \pi^\sigma_{-i}(z)$. The algorithm iterates over $i \in N$ and for each does a post-order depth-first traversal of the game tree, sampling actions at each history h where $P(h) \neq i$. In games where information sets are composed of histories solely due to the outcome of chance events being hidden from some players (*e.g.,* Bluff, Poker), then storing the choice of action sampled by $P(h) \neq i$ is unnecessary; due to perfect recall the algorithm will never visit more than one history from the same information set during this traversal. In the general case, some actions taken by player i may only be partly revealed (*e.g.,* Goofspiel, Phantom Tic-Tac-Toe), and so multiple histories in information sets belonging to $-i$ may be visited, therefore the choice of action sampled must be stored to ensure that the deterministic strategy sampled by the opponent remains consistent over all visited histories. For each visited information set belonging to i, the sampled counterfactual regrets are computed (and added to the total regrets).

When using external sampling and updating I belonging to player i and $z[I]a$ is a prefix of z, then $\pi^\sigma_{-i}(z[I], z) = \pi^\sigma_{-i}(z[I]a, z)$ since a is taken by i, not the opponent. Also note that $q(z) = \pi^\sigma_{-i}(z)$. So, the regret works out to be:

$$\begin{aligned}
\tilde{r}(I, a) &= \tilde{v}_i(\sigma^t_{(I\to a)}, I) - \tilde{v}_i(\sigma^t, I) && (4.16)\\
&= \sum_{z \in Q \cap Z_I} \frac{u_i(z)\pi^\sigma_{-i}(z[I])}{q(z)} \left(\pi^\sigma(z[I]a, z) - \pi^\sigma(z[I], z)\right) && (4.17)\\
&= \sum_{z \in Q \cap Z_I} \frac{u_i(z)\pi^\sigma_{-i}(z[I])\pi^\sigma_{-i}(z[I], z)}{q(z)} \left(\frac{\pi^\sigma(z[I]a, z) - \pi^\sigma(z[I], z)}{\pi^\sigma_{-i}(z[I], z)}\right) && (4.18)\\
&= \sum_{z \in Q \cap Z_I} \frac{u_i(z)\pi^\sigma_{-i}(z)}{q(z)} \left(\pi^\sigma_i(z[I]a, z) - \pi^\sigma_i(z[I], z)\right) && (4.19)\\
&= \sum_{z \in Q \cap Z_I} u_i(z) \left(\pi^\sigma_i(z[I]a, z) - \pi^\sigma_i(z[I], z)\right) && (4.20)
\end{aligned}$$

Since a single action is sampled at opponent information sets, the time required by each iteration is still exponential in the branching factor, but the exponent (which is equal to the depth in the case of Vanilla CFR), is divided by two (in a strictly alternating move game), and so each iteration takes roughly time proportional to the root of the size of the tree. If there are chance nodes, outcomes are also sampled and hence each iteration can take less than the root of the size of the tree.

4.4 Average Strategy Computation

Recall from Section 2.2.2, when using CFR the strategy approaching an approximate equilibrium is in fact $\bar{\sigma}^T$. If the goal is to compute an approximate equilibrium then, the profile that is retrieved once all the iterations are done is this average strategy profile.

The average strategy is defined, in Equation 2.14, as

$$\bar{\sigma_i}^T(I,a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I,a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)}, I \in \mathcal{I}_i$$

In vanilla CFR, only the numerator is accumulated for each action a at an information set I. When $\bar{\sigma_i}^T(I,a)$ is needed, the value maintained can be normalized over the values kept for all the other actions since $\sum_{a \in A(I)} \sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I,a) = \sum_{t=1}^{T} \pi_i^{\sigma^t}(I)$. The average strategy increment to action a at I is therefore $\pi_i^{\sigma^t}(I)\sigma^t(I,a)$; in practice, this increment is achieved via several smaller increments $\pi_i^{\sigma^t}(h)\sigma^t(I,a)$ over all $h \in I$. That is, the recursive traversal is over the game tree and not the information set tree. Since a full pass is done every iteration, the average strategy is computed exactly.

Computing the average strategy in MCCFR is generally less clear because only some of the information sets are visited each iteration. However, the computation of the average strategy at an information set should include how a player acted there over all T iterations. There are several approaches to solve this problem, which we describe below. Unlike in Vanilla CFR and chance-sampled CFR, the average strategy updates are applied on the opponent's turns to enforce the unbiasedness of the update to the average strategy.

One way to solve this problem is to have a counter at each information set c_I initially set to 0. When h is sampled and $h \in I$, the sum maintained for each action a is incremented by $(t - c_I)\pi_i^{\sigma^t}(h)\sigma^t(I,a)$ and then c_I is set to t. In effect, this weights the usual update to the average strategy by $(t - c_I)$, the number of time steps since we have last seen I. This is equivalent to performing $(t - c_I)$ updates to the average all at once, assuming the player had played the recently-computed strategy for all $(t - c_I)$ iterations since it was last sampled. Doing this is incorrect, as the increment to the average strategy should be

the sum of terms $\pi_i^{\sigma^t}(I)$ over iterations $\{t, t+1, \cdots\}$. We call this method **optimistic averaging**. There are several disadvantages to this method. The first, as noted above, is that it is only a heuristic approximation of the proper update. The second is that an extra variable is required increasing the amount of memory needed by $O(\mathcal{I})$. The third is that when the computation is finished and $\bar{\sigma}^T$ is needed, many of the information sets may not be up-to-date ($c_I < t$); therefore, when retrieving the average strategy, one final average strategy update is required at each information set.

Algorithm 3 Alternating Outcome Sampling with Opt. Avg. and ϵ-on-policy Exploration

Initialize: $\forall I \in \mathcal{I} : c_I \leftarrow 0$
Initialize: $\forall I \in \mathcal{I}, \forall a \in A(I) : r_I[a] \leftarrow s_I[a] \leftarrow 0$
OutcomeSampling(h, i, t, π_i, π_{-i}, s):
  if $h \in Z$ **then return** $(u_i(h)/s, 1)$
  if $P(h) = c$ **then** sample a' and **return** OutcomeSampling(ha', i, t, π_i, π_{-i}, s)
  Let I be the information set containing h
  $\sigma(I) \leftarrow$ RegretMatching(r_I)
  Let $\sigma'(I)$ be a sampling distribution at I
  if $P(I) = i$ **then** $\sigma'(I) \leftarrow \epsilon \cdot \text{Unif}(I) + (1-\epsilon)\sigma(I)$
  else $\sigma'(I) \leftarrow \sigma(I)$
  Sample an action a' with probability $\sigma'(I, a)$
  if $P(I) = i$ **then**
    $(u, \pi_{\text{tail}}) \leftarrow$ OutcomeSampling(ha', i, t, $\pi_i \cdot \sigma(I, a)$, π_{-i}, $s \cdot \sigma'(I, a)$)
    for $a \in A(I)$ **do**
      $W \leftarrow u \cdot \pi_{-i}$
      Compute $\tilde{r}(I, a)$ from Equation 4.12 **if** $a = a'$ **else** Equation 4.15
      $r_I[a] \leftarrow r_I[a] + \tilde{r}(I, a)$
  else
    $(u, \pi_{\text{tail}}) \leftarrow$ OutcomeSampling(ha', i, t, π_i, $\pi_{-i} \cdot \sigma(I, a)$, $s \cdot \sigma'(I, a)$)
    for $a \in A(I)$ **do**
      $s_I[a] \leftarrow s_I[a] + (t - c_I) \cdot \pi_{-i} \cdot \sigma(I, a)$
      $c_I \leftarrow t$
  return $(u, \pi_{\text{tail}} \cdot \sigma(I, a))$

One traversal of outcome sampling MCCFR with optimistic averaging for a single player i is presented in Algorithm 3. The h, t, and i parameters represent the current prefix, time step, and update player. The s parameter represents the product of the sampling probabilities up to h. The π_i and π_{-i} represent reach probabilities for each player. Unif(I) is the uniform distribution that assigns probability $1/|A(I)|$ to all actions a.

Another averaging method is to simply boost the magnitude of the update to the average strategy by the inverse probability of sampling the current history $\frac{1}{q(h)}$. For example, in outcome sampling using epsilon-on-policy exploration, this term is $\frac{1}{\pi^{\sigma'}(h)}$ whereas the term

Algorithm 4 External Sampling with Stochastically-Weighted Averaging

Initialize: $\forall I \in \mathcal{I}, \forall a \in A(I) : r_I[a] \leftarrow s_I[a] \leftarrow 0$
ExternalSampling(h, i):
  if $h \in Z$ **then return** $u_i(h)$
  if $P(h) = c$ **then** sample a' and **return** ExternalSampling(ha', i)
  Let I be the information set containing h
  $\sigma(I) \leftarrow$ RegretMatching(r_I)
  if $P(I) = i$ **then**
    Let u be an array indexed by actions and $u_\sigma \leftarrow 0$
    for $a \in A(I)$ **do**
      $u[a] \leftarrow$ ExternalSampling(ha, i)
      $u_\sigma \leftarrow u_\sigma + \sigma(I, a) \cdot u[a]$
    for $a \in A(I)$ **do**
      By Equation 4.20, compute $\tilde{r}(I, a) \leftarrow u[a] - u_\sigma$
      $r_I[a] \leftarrow r_I[a] + \tilde{r}(I, a)$
    return u_σ
  else
    Sample action a' from $\sigma(I)$
    $u \leftarrow$ ExternalSampling(ha', i)
    for $a \in A(I)$ **do**
      $s_I[a] \leftarrow s_I[a] + \sigma(I, a)$
    return u

is $\frac{1}{\pi^{\sigma^t}_{-i}(h)}$ in external sampling. Similar to optimistic averaging, this effectively applies the increment a number of times, but the number of times is determined by the current sampling strategy. This method attempts to correct the frequency of the updates back to the uniform distribution. We call this method **stochastically-weighted averaging**. One problem with this method is that the magnitude of the updates may have a large variance. When the likelihood of sampling an event is very small, the update has large weight; any error in the update will get magnified and may dominate over any future updates. However, this increment is an unbiased estimate of the true increment for the same reason that counterfactual value is an unbiased estimate of the true counterfactual value: effectively, it is an application of importance sampling to the average strategy update. Algorithm 4 presents external sampling with stochastically-weighted averaging. One benefit of using stochastically-weighted averaging in external sampling is that the reach probabilities no longer need to be propagated down the tree since they cancel out in both the regret update and average strategy update, simplifying the presentation of the algorithm.

Where the previous two methods are heuristic methods that attempt to fix the problem of proper averaging, there is a correct way to compute the average strategy. The method involves some extra book-keeping at each parent information set. This averaging method

Algorithm 5 Outcome sampling with lazy-weighted averaging and ϵ-on-policy exploration

Initialize: $\forall I \in \mathcal{I}, \forall a \in A(I) : w(I,a) \leftarrow 0$
OSampLWA(h, i, t, π_i, π_{-i}, s, w_1, w_2):
  if $h \in Z$ **then return** $(u_i(h)/s, 1)$
  if $P(h) = c$ **then** sample a' and **return** OSampLWA(ha', i, t, π_i, π_{-i}, s, w_1, w_2)
  Let I be the information set containing h
  $\sigma(I) \leftarrow$ RegretMatching(r_I)
  Let $\sigma'(I)$ be a sampling distribution at I
  if $P(h) = i$ **then** $\sigma'(I) \leftarrow \epsilon \cdot \text{Unif}(I) + (1-\epsilon)\sigma(I)$
  else $\sigma'(I) \leftarrow \sigma(I)$
  Sample an action a' with probability $\sigma'(I,a)$
  Let $w_1' \leftarrow w_1$
  Let $w_2' \leftarrow w_2$
  if $P(h) = i$ **then**
    $w_i' \leftarrow w_i \cdot \sigma(I,a')$
    $w_i' \leftarrow w_i' + w(I,a')$
    $(u, \pi_{\text{tail}}) \leftarrow$ OSampLWA(ha', i, t, $\pi_i \cdot \sigma(I,a)$, π_{-i}, $s \cdot \sigma'(I,a)$, w_1', w_2')
    for $a \in A(I)$ **do**
      $W \leftarrow u \cdot \pi_{-i}$
      Compute $\tilde{r}(I,a)$ from Equation 4.12 **if** $a = a'$ **else** Equation 4.15
      $r_I[a] \leftarrow r_I[a] + \tilde{r}(I,a)$
    for $a \in A(I)$ **do**
      $s_I[a] \leftarrow s_I[a] + (w_i + \pi_i) \cdot \sigma(I,a)$
      if $a = a'$ **then** $w(I,a) \leftarrow 0$
      else $w(I,a) \leftarrow w(I,a) + (w_i + \pi_i) \cdot \sigma(I,a)$
  else
    $w_{-i}' \leftarrow w_{-i} \cdot \sigma(I,a')$
    $w_{-i}' \leftarrow w_{-i}' + w(I,a')$
    $(u, \pi_{\text{tail}}) \leftarrow$ OSampLWA(ha', i, t, π_i, $\pi_{-i} \cdot \sigma(I,a)$, $s \cdot \sigma'(I,a)$, w_1', w_2')
    for $a \in A(I)$ **do**
      $s_I[a] \leftarrow s_I[a] + (w_{-i} + \pi_{-i}) \cdot \sigma(I,a)$
      if $a = a'$ **then** $w(I,a) \leftarrow 0$
      else $w(I,a) \leftarrow w(I,a) + (w_{-i} + \pi_{-i}) \cdot \sigma(I,a)$
  return $(u, \pi_{\text{tail}} \cdot \sigma(I,a))$

is presented with outcome sampling in Algorithm 5. The main idea is to maintain weights $w(I, a)$ at each (I, a) pair that keep track of a sum of reach products. These weights are initialized to 0 and are updated whenever the probability $\sigma^t(I, a)$ changes, or the node is visited with non-zero weight passed down from above. The value $w(I, a)$ represents a sum $\sum_{t \in \text{NotSampled}(T,I,a)} \pi_i^t \cdot \sigma_i^t(I, a)$ where $P(I) = i$ and NotSampled(T, I, a) is a subset of time steps $\{t_\perp + 1, t_\perp + 2, \cdots, T\}$ where T is the current iteration and action a was *not* sampled at I since the last time the average strategy was updated $t_\perp$. Effectively, this weight accumulates the portions of the reach likelihoods over time, and when action a is sampled, the sum is pushed down to the children. This sum of products is missing the reach probabilities from below, so these are multiplied into the terms on line 26; upon the return of the recursive call $w(I, a)$ is set back to zero on lines 23 and 31 to indicate that the accumulating reach probabilities have been passed forward and the process starts over. The weights essentially annotate changes that still need to be applied to the average strategy below (I, a) and will be pushed forward the next time a is sampled at I. We call this method **lazy-weighted averaging**. For the same reason as optimistic averaging, this method leaves information sets that have not been visited in a while out-of-date, and requires a similar final update to the average strategy to be applied after the last iteration. More importantly, more memory is required to store the weights at each (I, a). Since a weight is required for each (I, a), using this averaging scheme can increase the required memory by 50% since only two values are required for each (I, a) pair: a cumulative regret $r_I[a]$, and an average strategy total $s_I[a]$.

4.5 Regret Bounds and Convergence Rates

In this section, we give the regret bounds for MCCFR algorithms. Unlike the original CFR bounds, we will show a probabilistic bound on that holds with probability $1 - p$.

First, we need to return to the simplest form of **regret-matching**. Recall the setup of this problem from Section 2.2.2. Let A be a finite set of actions and $u^1, u^2, \cdots, u^T$ be any sequence where $u^t : A \to \mathbb{R}$ and $\max_{a,b \in A} |u^t(a) - u^t(b)| \leq \Delta^t$ is the payoff range at time t. We will define the cumulative regret of not playing with action a over all the trials similarly to Equation 2.10:

$$R^T(a) = \sum_{t=1}^{T} r^t(a) \text{ where } r^t(a) = u^t(a) - \sum_{b \in A} \sigma^t(b) u^t(b),$$

where σ^t is a mixed strategy used at time t. Recall $x^+ = \max(0, x)$ and define $R_{sum}^{T,+} =$

$\sum_{a \in A} R^{T,+}(a)$. Regret-matching is a procedure that assigns:

$$\sigma^{T+1}(a) = \begin{cases} \frac{R^{T,+}(a)}{R^{T,+}_{sum}} & \text{if } R^{T,+}_{sum} > 0; \\ 1/|A| & \text{otherwise} \end{cases}$$

We can refer to the average regret $\bar{R}^{T,+}(a) = R^{T,+}(a)/T$. If $\bar{R}$ is replaced into the equation above, there is no effect since the values are normalized and $T > 0$, but we require this notion for the proofs.

Theorem 2. *When regret-matching is used:*

$$\sum_{a \in A} (\bar{R}^{T,+}(a))^2 \leq \frac{|A|}{T^2} \sum_{t=1}^{T} (\Delta^t)^2.$$

The proof and associated lemmas are given in Appendix A.1.1. Theorem 2 gives a general bound on the regret of regret-matching that we will use to bound the regret at each information set. In particular, we will use the following:

Corollary 1. *When regret-matching is used, then* $\max_{a \in A} \bar{R}_i^{T,+}(a) \leq \frac{\sqrt{|A| \sum_{t=1}^{T} (\Delta^t)^2}}{T}$.

Proof. When $\max_{a \in A} \bar{R}^{T,+}(a)$ is non-positive, the result is trivial. Otherwise, there is at least one positive value, and:

$$\begin{aligned} \left(\max_{a \in A} \bar{R}^{T,+}(a) \right)^2 &= \max_{a \in A} \left(\bar{R}^{T,+}(a) \right)^2 \\ &\leq \sum_{a \in A} \left(\bar{R}^{T,+}(a) \right)^2 \\ &\leq \frac{1}{T^2} \sum_{t=1}^{T} |A| (\Delta^t)^2 \quad \text{by Theorem 2.} \end{aligned}$$

Now, we move $|A|$ out of the sum and take the square root of both sides. □

We now present four lemmas which will be used to support the main theorem below.

Lemma 2 (A Variant of Markov's Inequality)**.** *For any random variable X:*

$$\Pr[|X| \geq k \sqrt{\mathbb{E}[X^2]}] \leq \frac{1}{k^2}.$$

Proof. Markov's property states, if Y is always non-negative, then $\Pr(Y \geq j\mathbb{E}[Y]) \leq \frac{1}{j}$. By setting $Y = X^2$,

$$\Pr(X^2 \geq j\mathbb{E}[X^2]) \leq \frac{1}{j} \Rightarrow \Pr(|X| \geq \sqrt{j\mathbb{E}[X^2]}) \leq \frac{1}{j} \Rightarrow \Pr(|X| \geq k\sqrt{\mathbb{E}[X^2]}) \leq \frac{1}{k^2},$$

where the last step is obtained by setting $k = \sqrt{j}$. □

Lemma 3. *If $a_1 \ldots, a_n$ are non-negative real numbers in the interval $[0, 1]$ where $\sum_{i=1}^{n} a_i = S$, then $\sum_{i=1}^{n} (a_i)^2 \leq S$.*

Proof. The proof is trivial since $a_i \in [0, 1]$, then $\forall a_i, (a_i)^2 \leq a_i$. □

Lemma 4. *If $b_1 \ldots, b_n$ are non-negative real numbers where $\sum_{i=1}^{n} b_i^2 = S$, then $\sum_{i=1}^{n} b_i \leq \sqrt{Sn}$.*

Proof. First note that $(b_i - b_j)^2 \geq 0 \Rightarrow b_i^2 + b_j^2 \geq 2b_i b_j$. Now,

$$\left(\sum_{i=1}^{n} b_i\right)^2 = \sum_{i=1}^{n} b_i^2 + \sum_{i<j} 2b_i b_j \leq \sum_{i=1}^{n} b_i^2 + \sum_{i<j} (b_i^2 + b_j^2) = n \sum_{i=1}^{n} b_i^2$$

Taking the root of both sides gives the result. □

Lemma 5. *If $a_1 \ldots, a_n$ are non-negative real numbers where $\sum_{i=1}^{n} a_i = S$, then $\sum_{i=1}^{n} \sqrt{a_i} \leq \sqrt{Sn}$.*

Proof. Let $b_i = \sqrt{a_i}$ and apply Lemma 4. □

Lemma 6. *For any two reals $x, y \in \mathbb{R}$: $(x^+ - y^+)^2 \leq (x - y)^2$.*

Proof. First, a function $f : \Re \to \Re$ is said the be $L-$Lipschitz if $|f(x) - f(y)| \leq L|x - y|$. The function $f(x) = (x)^+$ is $1-$Lipschitz since it is equal to 0 when x is negative, and its growth rate is equal to 1 afterward. Therefore $|(x)^+ - (y)^+| \leq |x - y| \leq (x - y)^2$. □

We are almost ready to present the main theorem. Before doing so, we need a few more definitions. Recall the definition of a history h: it is a sequence of actions from the start of the game. Suppose we are playing Bluff(1,1), one sequence may be $h =$ (chance chooses ⚂ for player 1, chance chooses ⚃ for player 2, player 1 bids 1-3, player 2 bids 1-4, player 1 bids 1-5, player 2 bids 2-4), or compactly $h =$ (⚂, ⚃, 1-3, 1-4, 1-5, 2-4). Define a subsequence $\vec{a}_i(h)$ to be the sequence of actions taken by player i in the order they were taken in h, so that $\vec{a}_1(h) =$ (1-3, 1-5). We can simply refer to one such sequence belonging to player i, $\vec{a}_i \in \vec{A}_i = \{\vec{a}_i(h) : h \in H_i\}$. Denote $\mathcal{I}_i(\vec{a}_i)$ to be the set of information sets where player i's action sequence is $\vec{a}_i$, and the set of them to be $\mathcal{B}_i = \{\mathcal{I}_i(\vec{a}_i) : \vec{a}_i \in \vec{A}_i\}$. Note that $\mathcal{B}_i$ is a partition of $\mathcal{I}_i$. An element $B \in \mathcal{B}_i$ is a set of information sets that could occur given a particular subsequence $\vec{a}$, such as all the information sets I where player 1 bid 1-3 and then bid 1-5. Finally define $M_i = \sum_{B \in \mathcal{B}_i} \sqrt{|B|}$; this value tells us something about the size and structure of the game and is bounded by $\sqrt{|\mathcal{I}_i|} \leq M_i \leq |\mathcal{I}_i|$. For example, in a single-player game without chance nodes, each B is a singleton due to perfect recall, so

$M_i = |\mathcal{I}_i|$. Consider Figure 4.1; this game has $M_2 = \sqrt{|\mathcal{I}_i|}$ since there a single B with $|B| = 4$.

We now present the general MCCFR theorem.

Theorem 3. *For any $p \in (0, 1]$, when using any algorithm in the MCCFR family such that for all $Q \in \mathcal{Q}$ and $B \in \mathcal{B}_i$,*

$$\sum_{I \in B} \left(\sum_{z \in Q \cap Z_I} \frac{\pi^\sigma(z[I], z)\pi^\sigma_{-i}(z[I])}{q(z)} \right)^2 \leq \frac{1}{\delta^2}$$

where $\delta \leq 1$, then with probability at least $1 - p$, average overall regret is bounded by,

$$\bar{R}_i^T \leq \left(M_i + \frac{\sqrt{2|\mathcal{I}_i||\mathcal{B}_i|}}{\sqrt{p}} \right) \left(\frac{1}{\delta} \right) \frac{\Delta_{u,i}\sqrt{|A_i|}}{\sqrt{T}}.$$

Proof. Recall the definitions of counterfactual regret $r^t(I, a)$ and sampled counterfactual regret at time t from Equations 4.1 and 4.8. Now, define the cumulative immediate counterfactual regret and its sampled equivalent:

$$R_i^T(I) = \max_{a \in A(I)} \sum_{t=1}^{T} r_i^t(I, a) \qquad R_i^{T,+}(I) = \left(\max_{a \in A(I)} \sum_{t=1}^{T} r_i^t(I, a) \right)^+$$

$$\tilde{R}_i^T(I) = \max_{a \in A(I)} \sum_{t=1}^{T} \tilde{r}_i^t(I, a) \qquad \tilde{R}_i^{T,+}(I) = \left(\max_{a \in A(I)} \sum_{t=1}^{T} \tilde{r}_i^t(I, a) \right)^+$$

Let $Q_t \in \mathcal{Q}$ be the block sampled at time t. Note that we can bound the difference between two sampled counterfactual values for information set I at time t by,

$$\left(\tilde{v}_i(\sigma^t_{(I \to a)}, I) - \tilde{v}_i(\sigma^t, I) \right) \leq \Delta^t_{u,i}(I) \equiv \Delta_{u,i} \sum_{z \in Q_t \cap Z_I} \frac{\pi^\sigma(z[I], z)\pi^\sigma_{-i}(z[I])}{q(z)}$$

where $\Delta_{u,i} = \max_{z \in Z} u_i(z) - \min_{z \in Z} u_i(z)$ is the largest difference of the utility between any two leaves. By the condition stated in the theorem, we have:

$$\sum_{I \in B} \Delta^t_{u,i}(I)^2 \leq \frac{\Delta^2_{u,i}}{\delta^2}. \tag{4.21}$$

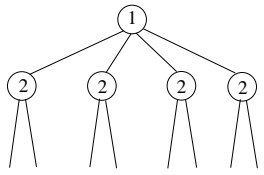


Figure 4.1: A game with $M_2 = \sqrt{|\mathcal{I}_2|}$.

We now apply Corollary 1 to bound the sampled cumulative regret at I:

$$\frac{\tilde{R}_i^{T,+}(I)}{T} \leq \frac{\sqrt{|A(I)| \sum_{t=1}^{T} (\Delta_{u,i}^t(I))^2}}{T} \tag{4.22}$$

Given some part $B \in \mathcal{B}_i$, we can sum over all I in B. Define $A(B) = \max_{I \in B} A(I)$:

$$\sum_{I \in B} \frac{\tilde{R}_i^{T,+}(I)}{T} \leq \frac{\sqrt{|B||A(B)| \sum_{I \in B} \sum_{t=1}^{T} (\Delta_{u,i}^t(I))^2}}{T} \quad \text{by Lemma 5} \tag{4.23}$$

$$\leq \frac{\sqrt{|B||A(B)| \sum_{t=1}^{T} \sum_{I \in B} (\Delta_{u,i}^t(I))^2}}{T} \tag{4.24}$$

$$\leq \frac{\sqrt{|B||A(B)| \sum_{t=1}^{T} \Delta_{u,i}^2/\delta^2}}{T} \quad \text{by Equation 4.21} \tag{4.25}$$

$$\leq \frac{\Delta_{u,i} \sqrt{|B||A(B)|}}{\delta \sqrt{T}} \tag{4.26}$$

Now we can bound the average over all information sets by

$$\sum_{I \in \mathcal{I}_i} \frac{\tilde{R}_i^{T,+}(I)}{T} = \sum_{B \in \mathcal{B}_i} \sum_{I \in B} \frac{\tilde{R}_i^{T,+}(I)}{T} \quad \text{since } \mathcal{B}_i \text{ is a partition of } \mathcal{I}_i \tag{4.27}$$

$$\leq \sum_{B \in \mathcal{B}_i} \frac{\Delta_{u,i} \sqrt{|B||A(B)|}}{\delta \sqrt{T}} \quad \text{by Equation 4.26} \tag{4.28}$$

$$\leq \frac{\Delta_{u,i} \sqrt{|A_i|} \sum_{B \in \mathcal{B}_i} \sqrt{|B|}}{\delta \sqrt{T}} \quad \text{since } \sqrt{|A_i|} \geq \sqrt{|A(B)|} \tag{4.29}$$

$$\leq \frac{\Delta_{u,i} M_i \sqrt{|A_i|}}{\delta \sqrt{T}} \quad \text{by definition of } M_i \tag{4.30}$$

We will now prove that $R_i^{T,+}(I)$ and $\tilde{R}_i^{T,+}(I)$ are similar which lets us apply the CFR theorem [124, Theorem 3] allowing us to bound the current overall regret by the per-information set regret. This last portion is tricky. Since the algorithm is randomized, we cannot guarantee that every information set is reached, let alone that it has converged. Therefore, instead of proving a bound on the absolute difference of R and $\tilde{R}$, we focus on proving a probabilistic connection.

In particular, we will bound the expected squared difference between $\sum_{I \in \mathcal{I}_i} \frac{R_i^{T,+}(I)}{T}$ and $\sum_{I \in \mathcal{I}_i} \frac{\tilde{R}_i^{T,+}(I)}{T}$ in order to prove that they are close in expectation, and then use a variant of Markov's inequality to bound the absolute value. We begin by focusing on the similarity of the counterfactual regret ($\frac{R_i^{T,+}(I)}{T}$ and $\frac{\tilde{R}_i^{T,+}(I)}{T}$) in every node, by focusing on the similarity of the counterfactual regret of a particular action at a particular time ($r_i^t(I,a)$ and $\tilde{r}_i^t(I,a)$). By the Lemma 1, we know that $\mathbb{E}[r_i^t(I,a) - \tilde{r}_i^t(I,a)] = 0$.

For $I \in \mathcal{I}_i$,

$$\begin{aligned}
\left(R_i^{T,+}(I) - \tilde{R}_i^{T,+}(I)\right)^2 &= \left(\left(\max_{a\in A(I)} \sum_{t=1}^{T} r_i^t(I,a)\right)^+ - \left(\max_{a\in A(I)} \sum_{t=1}^{T} \tilde{r}_i^t(I,a)\right)^+\right)^2 \\
&\leq \left(\max_{a\in A(I)} \sum_{t=1}^{T} r_i^t(I,a) - \max_{a\in A(I)} \sum_{t=1}^{T} \tilde{r}_i^t(I,a)\right)^2 \qquad \text{by Lemma 6} \\
&\leq \left(\max_{a\in A(I)} \sum_{t=1}^{T} \left(r_i^t(I,a) - \tilde{r}_i^t(I,a)\right)\right)^2 \\
&\leq \max_{a\in A(I)} \left(\sum_{t=1}^{T} \left(r_i^t(I,a) - \tilde{r}_i^t(I,a)\right)\right)^2
\end{aligned}$$

$$\begin{aligned}
&\leq \sum_{a\in A(I)} \left[\sum_{t=1}^{T} \left(r_i^t(I,a) - \tilde{r}_i^t(I,a)\right)^2 \right. \\
&\quad \left. +2\sum_{t=1}^{T}\sum_{t'=t+1}^{T} \left(r_i^t(I,a) - \tilde{r}_i^t(I,a)\right)\left(r_i^{t'}(I,a) - \tilde{r}_i^{t'}(I,a)\right)\right]. \qquad (4.31)
\end{aligned}$$

We now take the expectation of both sides. Note that

$$\begin{aligned}
&\mathbb{E}\left[\left(r_i^t(I,a) - \tilde{r}_i^t(I,a)\right)\left(r_i^{t'}(I,a) - \tilde{r}_i^{t'}(I,a)\right)\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\left(r_i^{t'}(I,a) - \tilde{r}_i^{t'}(I,a)\right) \mid r_i^t(I,a), \tilde{r}_i^t(I,a)\right]\left(r_i^t(I,a) - \tilde{r}_i^t(I,a)\right)\right]
\end{aligned}$$

and that $\mathbb{E}\left[\left(r_i^{t'}(I,a) - \tilde{r}_i^{t'}(I,a)\right) \mid r_i^t(I,a), \tilde{r}_i^t(I,a)\right] = 0$ since for $t' > t$, $\tilde{r}_i^{t'}$ is an unbiased estimate of $r_i^{t'}$ given $\sigma^{t'}$. Thus from equation (4.31), we have

$$\begin{aligned}
\mathbb{E}\left[\left(R_i^{T,+}(I) - \tilde{R}_i^{T,+}(I)\right)^2\right] &\leq \sum_{a\in A(I)} \sum_{t=1}^{T} \mathbb{E}\left[\left(r_i^t(I,a) - \tilde{r}_i^t(I,a)\right)^2\right] \\
&\leq \sum_{a\in A(I)} \sum_{t=1}^{T} \mathbb{E}\left[\left(r_i^t(I,a)\right)^2 + \left(\tilde{r}_i^t(I,a)\right)^2\right] \\
&\leq \sum_{a\in A(I)} \sum_{t=1}^{T} \left[\left(\pi_{-i}^{\sigma^t}(I)\right)^2 \Delta_i^2 + \left(\Delta_i^t(I)\right)^2\right]. \qquad (4.32)
\end{aligned}$$

We can now bound the expected sum of squared differences by

$$
\begin{aligned}
&\mathbb{E}\left[\left(\sum_{I\in\mathcal{I}_i}\left(R_i^{T,+}(I)-\tilde{R}_i^{T,+}(I)\right)\right)^2\right] \\
&\quad\le \mathbb{E}\left[\left(\sum_{I\in\mathcal{I}_i}\left|R_i^{T,+}(I)-\tilde{R}_i^{T,+}(I)\right|\right)^2\right] \\
&\quad\le \mathbb{E}\left[\left(\sqrt{|\mathcal{I}_i|\sum_{I\in\mathcal{I}_i}\left|\left(R_i^{T,+}(I)-\tilde{R}_i^{T,+}(I)\right)\right|^2}\right)^2\right] \quad \text{by Lemma 4} \\
&\quad= |\mathcal{I}_i|\sum_{I\in\mathcal{I}_i}\mathbb{E}\left[\left(R_i^{T,+}(I)-\tilde{R}_i^{T,+}(I)\right)^2\right] \\
&\quad\le |\mathcal{I}_i|\sum_{I\in\mathcal{I}_i}\sum_{a\in A(I)}\sum_{t=1}^{T}\left[\left(\pi_{-i}^{\sigma^t}(I)\right)^2\Delta_i^2+\left(\Delta_i^t(I)\right)^2\right] \quad \text{by equation (4.32)} \\
&\quad\le |\mathcal{I}_i||A_i|\sum_{B\in\mathcal{B}_i}\sum_{t=1}^{T}\left[\sum_{I\in B}\left(\pi_{-i}^{\sigma^t}(I)\right)^2\Delta_i^2+\sum_{I\in B}\left(\Delta_i^t(I)\right)^2\right]
\end{aligned}
$$

$$
\begin{aligned}
&\le |\mathcal{I}_i||A_i|\sum_{B\in\mathcal{B}_i}\sum_{t=1}^{T}\left[\Delta_i^2+\frac{\Delta_i^2}{\delta^2}\right] \quad \text{by Lemma 16 of [71] and equation 4.21} \\
&\le \frac{2|\mathcal{I}_i||A_i||\mathcal{B}_i|T\Delta_i^2}{\delta^2} \qquad (4.33)
\end{aligned}
$$

Finally, with probability $1-p$, we can bound the regret by

$$
\begin{aligned}
R_i^T &\le \sum_{I\in\mathcal{I}_i}R_i^{T,+}(I) \quad \text{by the original CFR Theorem 3 of [124]} \\
&= \sum_{I\in\mathcal{I}_i}\left(R_i^{T,+}(I)-\tilde{R}_i^{T,+}(I)+\tilde{R}_i^{T,+}(I)\right) \\
&\le \left|\sum_{I\in\mathcal{I}_i}\left(R_i^{T,+}(I)-\tilde{R}_i^{T,+}(I)\right)\right|+\sum_{I\in\mathcal{I}_i}\tilde{R}_i^{T,+}(I) \\
&\le \frac{1}{\sqrt{p}}\sqrt{\mathbb{E}\left[\left(\sum_{I\in\mathcal{I}_i}\left(R_i^{T,+}(I)-\tilde{R}_i^{T,+}(I)\right)\right)^2\right]}+\frac{\Delta_i M_i\sqrt{|A_i|T}}{\delta} \\
&\qquad\qquad \text{by Lemma 2 and equation (4.30)} \\
&\le \left(\frac{\sqrt{2|\mathcal{I}_i||\mathcal{B}_i|}}{\sqrt{p}}+M_i\right)\left(\frac{1}{\delta}\right)\Delta_i\sqrt{|A_i|T} \quad \text{by equation (4.33)} \qquad (4.34)
\end{aligned}
$$

Dividing both sides by T gives the result. □

We will finish this section by using the general MCCFR theorem to give the specific bounds for both outcome sampling and external sampling. A tighter bound for Vanilla CFR and

chance-sampled CFR can be obtained as well using these bounds based on M_i and $|\mathcal{B}_i|$; see [71, Section A.7] for the derivation.

4.5.1 Outcome Sampling Bound

Theorem 4. *For any $p \in (0, 1]$, when using outcome-sampling MCCFR where $\forall z \in Z$ either $\pi^{\sigma}_{-i}(z) = 0$ or $q(z) \geq \delta > 0$ at every time step, with probability $1 - p$, average overall regret is bounded by*

$$\bar{R}_i^T \leq \left(\frac{\sqrt{2|\mathcal{I}_i||\mathcal{B}_i|}}{\sqrt{p}} + M_i \right) \left(\frac{1}{\delta} \right) \frac{\Delta_{u,i}\sqrt{|A_i|}}{\sqrt{T}}$$

Proof. We simply need to show that,

$$\sum_{I \in B} \left(\sum_{z \in Q \cap Z_I} \frac{\pi^{\sigma}(z[I], z)\pi^{\sigma}_{-i}(z[I])}{q(z)} \right)^2 \leq \frac{1}{\delta^2}.$$

Note that for all $Q \in \mathcal{Q}$, $|Q| = 1$. Also note that for any $B \in \mathcal{B}_i$ there is at most one $I \in B$ such that $Q \cap Z_I \neq \emptyset$. This is because all the information sets in $Q \cap Z_I$ all have player i's action sequence of a different length, while all information sets in B have player i's action sequence being the same length. Therefore, only a single term of the inner sum is ever non-zero.

Now by our assumption, for all I and $z \in Z_I$ where $\pi^{\sigma}_{-i}(z) > 0$,

$$\frac{\pi^{\sigma}(z[I], z)\pi^{\sigma}_{-i}(z[I])}{q(z)} \leq \frac{1}{\delta}$$

as all the terms of the numerator are less than 1. So the one non-zero term is bounded by $1/\delta$ and so the overall sum of squares must be bounded by $1/\delta^2$. □

4.5.2 External Sampling Bound

Theorem 5. *For any $p \in (0, 1]$, when using external-sampling MCCFR, with probability at least $1 - p$, average overall regret is bounded by*

$$R_i^T \leq \left(\frac{\sqrt{2|\mathcal{I}_i||\mathcal{B}_i|}}{\sqrt{p}} + M_i \right) \frac{\Delta_{u,i}\sqrt{|A_i|}}{\sqrt{T}}. \tag{4.35}$$

Proof. We will simply show that,

$$\sum_{I \in B} \left(\sum_{z \in Q \cap Z_I} \frac{\pi^{\sigma}(z[I], z)\pi^{\sigma}_{-i}(z[I])}{q(z)} \right)^2 \leq 1 \tag{4.36}$$

Since $q(z) = \pi^{\sigma}_{-i}(z)$, we need to show,

$$\sum_{I \in B} \left(\sum_{z \in Q \cap Z_I} \pi^{\sigma}_i(z[I], z) \right)^2 \leq 1 \tag{4.37}$$

Let $\hat{\sigma}^t$ be a deterministic strategy profile sampled from σ^t where Q is the set of histories consistent with $\hat{\sigma}^t_{-i}$. So $Q \cap Z_I \neq \emptyset$ if and only if I is reachable with $\hat{\sigma}^t_{-i}$. By [71, Lemma 10], for all $B \in \mathcal{B}_i$ there is only one $I \in B$ that is reachable; name it I^*. Moreover, there is a unique history in I^* that is a prefix of all $z \in Q \cap Z_{I^*}$; name it h^*. So for all $z \in Q \cap Z_{I^*}$, $z[I^*] = h^*$. This is because $\hat{\sigma}^t_{-i}$ uniquely specifies the actions for all but player i and B uniquely specifies the actions for player i prior to reaching I^*.

Define ρ to be a strategy for all players (including chance) where $\rho_{j \neq i} = \hat{\sigma}_j$ but $\rho_i = \sigma_i$. Consider a $z \in Q \cap Z_I$. z must be reachable by $\hat{\sigma}_{-i}$, so $\pi^{\rho}_{-i}(z) = 1$. So

$$\sum_{z \in Q \cap Z_{I^*}} \pi^{\sigma}_i(z[I^*], z) = \sum_{z \in Q \cap Z_{I^*}} \pi^{\rho}_i(h^*, z) \tag{4.38}$$

$$= \sum_{z \in Q \cap Z_{I^*}} \pi^{\rho}(h^*, z) \tag{4.39}$$

$$\leq \sum_{z \in Z_{I^*}} \pi^{\rho}(h^*, z) \leq 1 \tag{4.40}$$

So,

$$\sum_{I \in B} \left(\sum_{z \in Q \cap Z_I} \pi^{\sigma}_i(z[I], z) \right)^2 \leq 1 \tag{4.41}$$

□

How do these bounds compare to the original CFR bound? If we assume a constant tolerance error probability $1 - p$, then the original CFR bound is roughly on the same order as the external sampling bound (Equation 4.35). This means that the number of iterations required to reach an ϵ-equilibrium are the same in both cases, except that in external sampling MCCFR we have this probability of error p. If we assume that one can tolerate an error rate of p, then we can treat this as a constant and so the bound is boosted by a constant number of iterations. However, the time spent per iteration in external sampling for a balanced game is roughly the square root of the time spent per CFR iteration. Therefore, external sampling does achieve an asymptotic improvement in convergence speed.

For the case of outcome sampling, there is the additional term $\frac{1}{\delta}$. The δ parameter is the smallest probability of sampling any one terminal history $z \in Z$. For $\frac{1}{\delta}$ to be maximized, one would choose $\delta = \frac{1}{|Z|}$. For a large game, this can be very small and so the bound for

outcome sampling can be largely affected. However, if total time is the quantity of interest: each iteration of outcome sampling is $|Z|$ times faster than an iteration of Vanilla CFR, so the requirement of $|Z|$ times as many iterations cancels out with the speedup term.

4.6 Empirical Evaluation

In this section, we present a number of experiments to determine the behavior of MCCFR in practice.

In all of the following experiments, we measure the convergence rate of the algorithm by computing the **exploitability** value described in Section 2.1 at certain stopping points during their execution. The exploitability value can be obtained by running a best response algorithm twice, one for each player. Suppose the game value (expected value for the first player if both players play an equilibrium) is v, and suppose we have a profile $\sigma = (\sigma_1, \sigma_2)$. The best response algorithm computes $\max_{\sigma_2' \in \Sigma_2} u_2(\sigma_1, \sigma_2') = -v + \epsilon_1$ and $\max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2) = v + \epsilon_2$. If σ is an equilibrium, then it must be the case that $\epsilon_1 = \epsilon_2 = 0$. Before reaching equilibrium, the exploitability value $\epsilon_\sigma = \epsilon_1 + \epsilon_2$ can be obtained by summing the best response values (as the value of the game v cancels in the sum). For games with entirely public actions, the best response algorithm can be computed using expectimax [102; 53]. When actions are hidden or only partially observable, the best response algorithm is slightly more complex. These best response algorithms are described in Appendix B.

As each sampling algorithm samples different portions of the tree, the time required by each iteration varies. Therefore, we chose not to measure convergence as a function of iterations. On the other hand, plotting the exploitability over time can be biased by specific implementation decisions. Therefore, we use the cumulative number of nodes touched as the value on the x-axes, which is equivalent to the total number of prefix histories enumerated over all iterations so far. For reference, a single-threaded implementation of external sampling written in C++ compiled using g++ version 4.6.3 in Ubuntu Linux visits about 3 million nodes per second on Bluff(1,1) on an Intel Core 2 Duo E6850 CPU at 3.00GHz.

4.6.1 Outcome Sampling with Varying Parameter Settings

We start by presenting experiments on outcome sampling with optimistic averaging, in two games: Bluff(1,1) and Goofspiel(6). Recall from Algorithm 3 that outcome sampling uses an ϵ-on-policy exploration strategy to choose which action to sample next: with probability $(1-\epsilon)$, a move is sampled from the current strategy $\sigma(I)$, otherwise a random action in $A(I)$

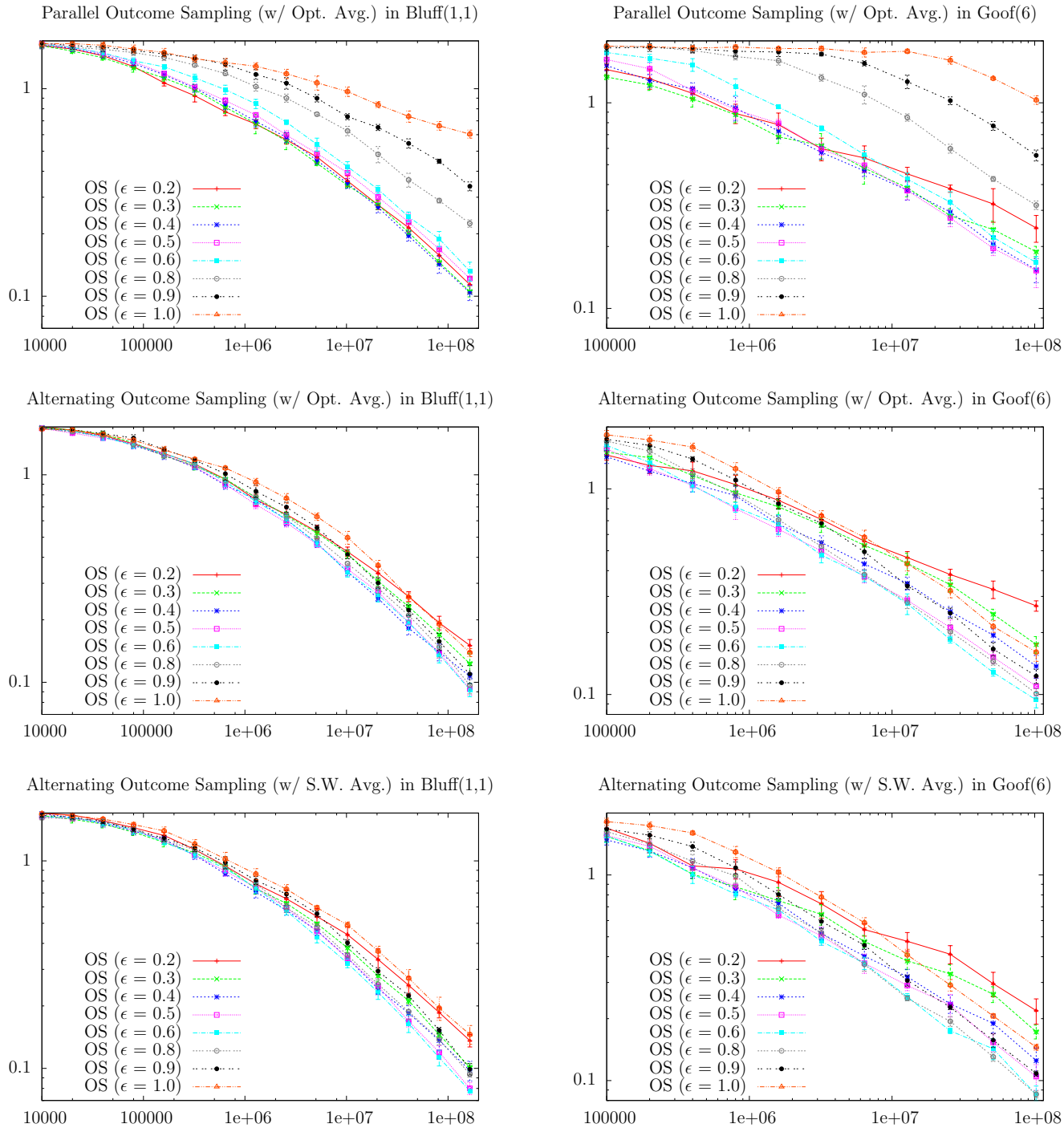


Figure 4.2: Outcome Sampling on Bluff(1,1) (left) and Goof(6) (right) with various parameter settings. The horizontal axes represent cumulative nodes touched and the vertical axes represents the exploitability value ϵ_σ. Logarithmic scales are used for both axes. Each line is an average over 5 runs with error bars representing 95% confidence intervals. Due to the very small width of the error bars, the dots on the graph may seem to have a different shape than those in the legend. Each Bluff graph uses the exact same vertical range; similarly, for the Goofspiel graphs.

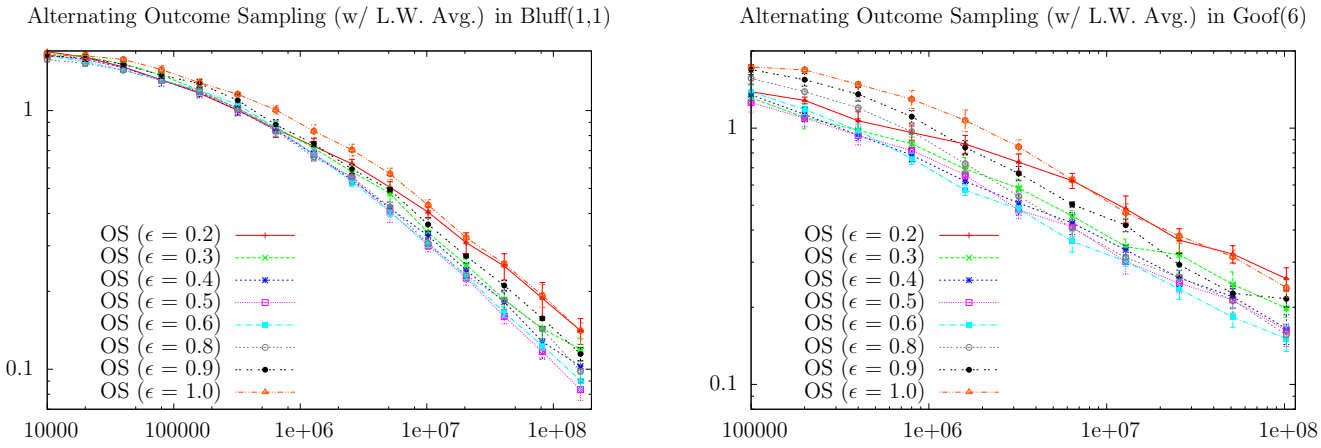


Figure 4.3: Outcome Sampling on Bluff(1,1) (left) and Goof(6) (right) using lazy-weighted averaging. The horizontal axes represent cumulative nodes touched and the vertical axes represents the exploitability value ϵ_σ. Logarithmic scales are used for both axes. Each line is an average over 5 runs with error bars representing 95% confidence intervals. Due to the very small width of the error bars, the dots on the graph may seem to have a different shape than those in the legend. Each Bluff graph uses the exact same vertical range as Figure 4.2; similarly, for the Goofspiel graphs.

is sampled. Recall from Section 4.2 that in outcome sampling one can re-use the sampled terminal history to update the sampled regrets of both player's information sets (the parallel form). Our initial goals are to find a reasonably good setting for the exploration constant ϵ, and compare the performance of the alternating form to the parallel form.

The resulting convergence rates are shown in Figure 4.2. Like the bound, the empirical exploitability tends to drop roughly proportional to an inverse square function with respect to time. Therefore, to compare and analyze convergence rates, we use logarithmic scale on both axes. On these graphs, a line with a steeper (more negative) slope than another indicates a faster convergence.

The first observation is that when using the parallel form, the convergence rate can be slow or erratic when $\epsilon > 0.6$. Even when using lower values of ϵ, the parallel form does not seem to be converging faster than the alternating form, which does not exhibit the irregularity when $\epsilon > 0.6$. We believe that the cause of this is due to the updates to the average strategy being too noisy due to too much exploration. Therefore, we conclude that the alternating form presented above is safer to use in practice. In both games, when using the alternating form, $0.5 \leq \epsilon \leq 0.8$ appears to be a good interval for this value, with $\epsilon = 0.6$ being a good choice overall. In addition, we ran alternating outcome sampling

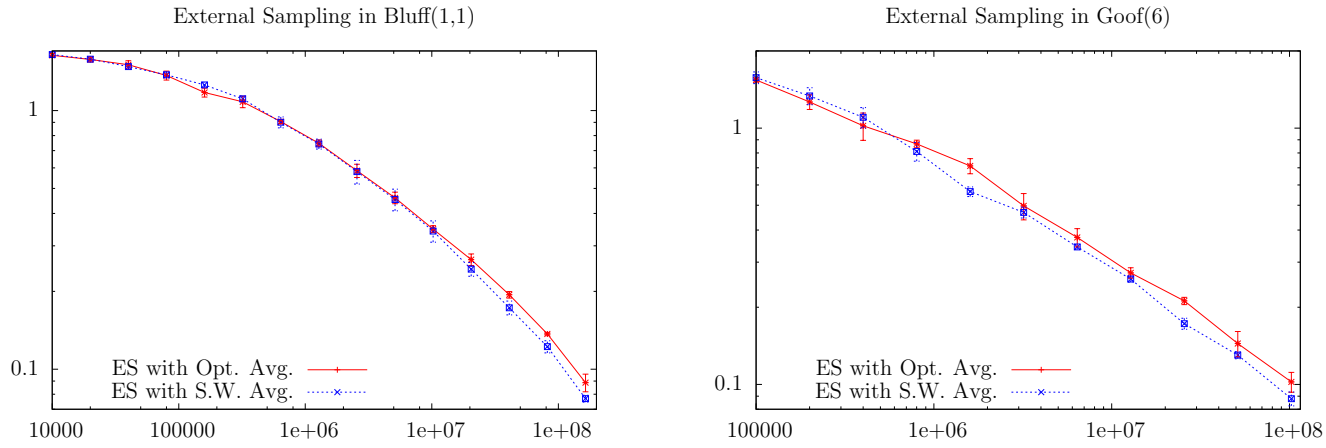


Figure 4.4: External Sampling on Bluff(1,1) (left) and Goof(6) (right) using optimistic and stochastically-weighted averaging. The horizontal axes represent cumulative nodes touched and the vertical axes represents the exploitability value ϵ_σ. Logarithmic scales are used for both axes. Each line is an average over 5 runs with error bars representing 95% confidence intervals. Due to the very small width of the error bars, the dots on the graph may seem to have a different shape than those in the legend. The Bluff graph uses the exact same vertical range as Figures 4.2 and 4.3; similarly, for the Goofspiel graphs.

on these two games with stochastically-weighted averaging, which has higher variance but shows a slight improvement over optimistic averaging in both games.

Finally, we ran alternating outcome sampling with lazy-weighted averaging; the results are shown in Figure 4.3. We see the convergence rates for lazy-weighted averaging are slower than stochastically-weighted averaging and, in the case of Goofspiel, slower than optimistic averaging; this might be due to the heuristic nature of the first two averaging schemes. When the average strategy is updated at an information set using optimistic averaging, the update player's reach strategy π_i is assumed to have been equal to the current one since the last update time c_I; so $(T - c_I)\pi_i^T$ could be a better (more informed) value than $\sum_{t=c_I}^{T} \pi_i^t$. As for stochastically-weighted averaging, since the updates to the average strategy are divided by $\sigma'(h)$, the magnitude of the updates are boosted by $\frac{1}{\sigma'(h)}$; therefore, when it is time to take the best response, these value estimates could be represent a value which is many iterations ahead of T. Overall, lazy-weighted averaging also requires additional storage and overhead in terms of computation time (to compute the weights and to patch the average strategy at the end); therefore, stochastically-weighted averaging appears to work best in practice on these games.

4.6.2 External Sampling with Different Averaging Schemes

The next simple experiment was run to determine which averaging scheme worked best with external sampling. The experiments were repeated using the same games as before. The results are shown in Figure 4.4. Since stochastically-weighted averaging leads to faster convergence in practice for both external and outcome sampling, requires less memory, and does not need a patch to be applied after the iterations are done, we conclude that it is the most practical averaging scheme of the three[2]

4.6.3 Convergence Rate Comparison in Large Games

The next set of experiments aims to compare the relative empirical performance of each sampling algorithm. For this set of experiments, we compare Vanilla CFR, chance-sampled CFR (where applicable), outcome sampling MCCFR, and external sampling MCCFR. These experiments are run on a number of significantly larger games: Bluff(2,1), One-Card Poker with a 500-card deck, Princess and Monster, Latent Tic-Tac-Toe, and Goofpsiel(7). The results are shown in Figure 4.5.

There are several observations that can be made about the empirical performance of MCCFR from these graphs. Generally, the MCCFR variants (including chance-sampling) outperform vanilla CFR. For example, in Goofspiel, both MCCFR variants require only a few million nodes to reach $\epsilon_\sigma < 0.5$ where CFR takes 2.5 billion nodes, three orders of magnitude more. In fact, external-sampling outperforms CFR by considerable margins in all of the games. Note that pruning optimization is key to vanilla CFR being at all practical in these games. For example, in Latent Tic-Tac-Toe the first iteration of CFR touches 39 million nodes and the next few iterations each touch only between 5 and 10 million nodes. This is because pruning is not possible in the first iteration. We believe this is due to dominated actions in the game. After one or two traversals, the players identify and eliminate dominated actions from their policies, allowing these subtrees to be pruned. This is especially apparent in Bluff where outcome sampling performs relatively poorly. We believe that this is due to the high proportion of bad actions available and the size of the corresponding subtrees below them; since outcome sampling constantly explores these actions always have a chance of being sampled, where even pruning with Vanilla CFR can avoid these. In certain cases such as One-Card Poker, Princess and Monster, and

[2]Members of the Computer Poker Research Group claim that across several independent implementations, after billions of iterations, external sampling using stochastically-weighted averaging convergences significantly faster than using optimistic averaging, in abstract versions of Two-Player Limit Texas Hold'em[55].

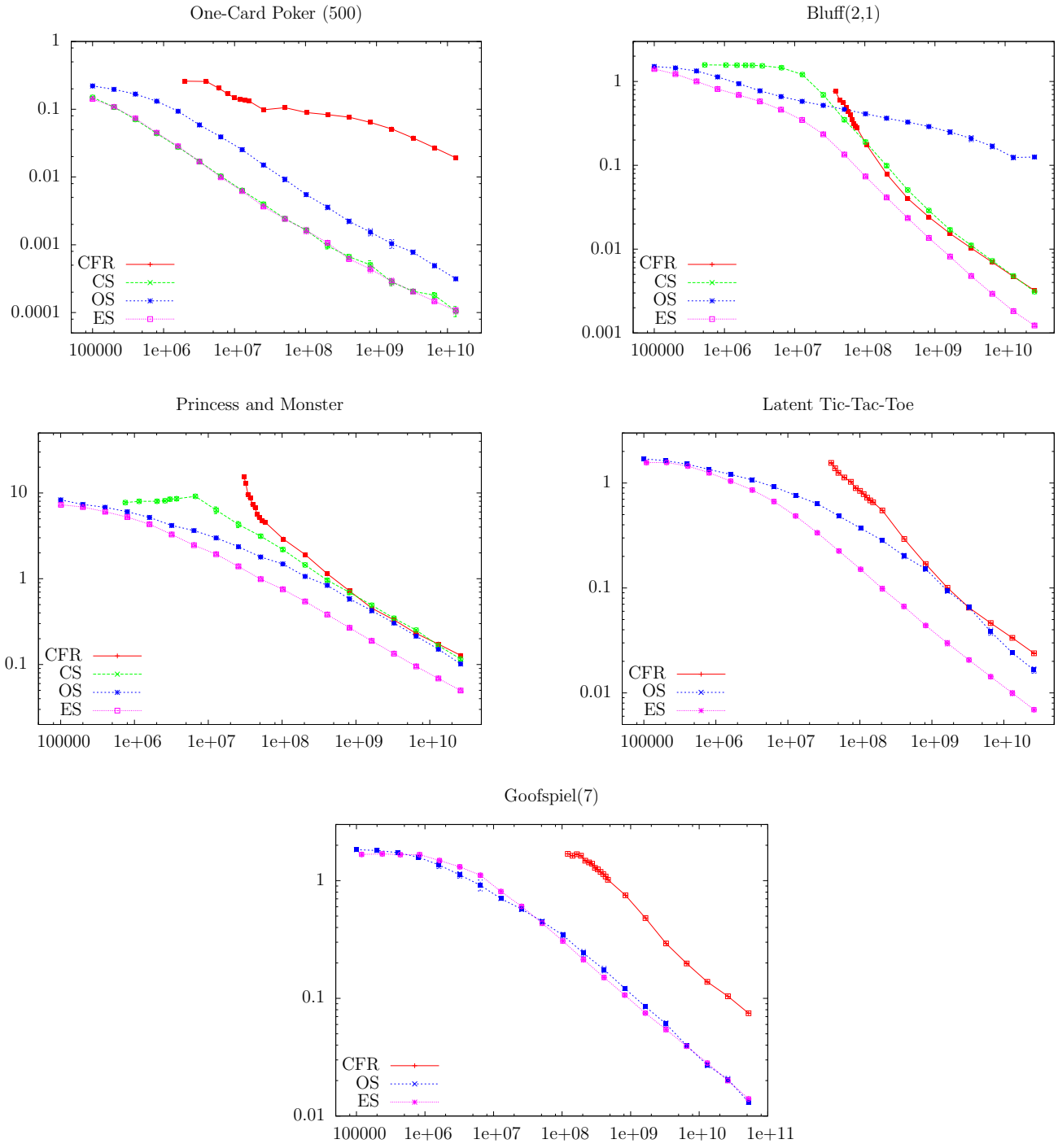


Figure 4.5: Comparison of CFR and MCCFR on a variety of games. The horizontal axes represent cumulative nodes touched and the vertical axes represents the exploitability value ϵ_σ. Logarithmic scales are used for both axes. Each line is an average over 5 runs with error bars representing 95% confidence intervals. Due to the very small width of the error bars, the dots on the graph may seem to have a different shape than those in the legend. CFR refers to Vanilla CFR while CS refers the chance-sampled CFR. Outcome sampling MCCFR (OS) and external sampling MCCFR (ES) both use stochastically-weighted averaging, and OS uses an exploration constant of $\epsilon = 0.6$.

Goofspiel, the eventual relative convergence rates (slopes) appear to be comparable, but the initial information gained at the start of the trials lead to better overall convergence rates in MCCFR. In games like One-Card Poker, where there are many combinations of chance outcomes that effectively lead to the same payoff, Vanilla CFR will converge particularly poorly, and the benefit of sampling is greater.

While generally external-sampling appears to be the best choice overall, outcome sampling has a slightly faster convergence rate in Goofspiel(7). This is likely due to the structure of the specific Goofspiel variant we chose. Since the point cards are arranged from highest down to lowest, the strategy to play in the first few turns is critical to having the most points in the game. Comparatively, outcome sampling will apply more updates at the top of the tree (per node touched) than any of the other CFR algorithms, so it may be learning the right strategy at these critical decision points early. This observation suggests a possible extension: a sampling scheme that is tailored to the game's structure that focuses on sampling important parts of the tree for faster convergence. The on-policy and ϵ-on-policy schemes we describe above do tend to focus on the areas that players will play in, but these policies are built by learning; injecting some amount of prior knowledge of the domain into the sampling scheme may help.

4.7 Discussion and Applicability of MCCFR

The idea of sampling parts of the tree was originally motivated by the success of chance-sampled CFR in Poker. In this chapter, we described a framework for sampling, but the questions remains: why sample at all? What is the benefit of sampling and when is it preferred to not sampling? Other than the general improvement shown in Section 4.5, there may be situations where there is simply not enough time available to to run a single iteration of Vanilla CFR. In this case, an MCCFR algorithm may have time time to run several iterations. As we saw in Section 4.6, MCCFR's short-term convergence speed is much better than CFR in practice. In the extreme case of real-time decision-making in imperfect information games, outcome sampling may be preferred since several iterations are likely possible and more time is spent learning about the actions at the root of the tree. Given some fixed longer amount of time, more iterations of MCCFR can be completed, and the sampling scheme gets progressively more informed as learning compounds faster since subsequent iterations make use of what was learned on past iterations.

Given a game or a particular problem, how does one choose the appropriate sampling

mechanism to use when employing an MCCFR algorithm? Many games have a high branching factor, but the relative importance of each action is quite different; often there are a number of bad actions that skilled players never play. In these cases, the sampling scheme can incorporate domain knowledge (sampling the bad moves less); also, using the sampling schemes that are a function of the current strategy (outcome sampling and external sampling), these bad moves are likely to be identified quickly. Since outcome sampling always explores, however, these actions may still be taken at later iterations. Since external sampling provides a better bound, and from results of experiments in Section 4.6, external sampling is the variant we expect to work better in practice (of the two) most often. On the other hand, the experiments in Goofspiel may provide evidence that outcome sampling will converge faster in games where the importance of each action decreases from root to leaf. Outcome sampling may also be expected to do well when the variance in the payoffs are low, since there is little detriment to exploring in these cases. Finally, in the imperfect information search setting, there is often limited search time, and so a slight modification of outcome sampling can be used that samples the current information set (the one given to the search algorithm describing the current situation) more often.

While we present regret-matching as the underlying regret minimizer at each information set, in principle any regret minimizer can be used in CFR and MCCFR. The convergence rates will of course differ from the ones we present here, but the convergence to an equilibrium will be preserved. In CFR and chance-sampled CFR, regret-matching is a particularly suitable choice because it assigns zero to actions with negative cumulative regret, leading to many opportunities to prune subtrees using the optimization described in Section 2.2.2. In MCCFR, there is substantially less benefit to pruning, since there are less instances where the optimization can be applied. Therefore, one straight-forward extension would be to use the Hedge algorithm, described in Section 2.2.2, for sampling actions at each information set. The reward signals would be the same sampled counterfactual values described here, but the strategy update would change.

Finally, we give a conjecture: ensuring that the current strategy σ^t is fully mixed (*i.e.,* $\forall I \in \mathcal{I}, a \in A(I) : \sigma^t(I,a) \neq 0$) at every iteration t (as guaranteed by a Hedge-based CFR algorithm) may lead to convergence to a sequential equilibrium and/or trembling-hand-perfect equilibrium.

Overall, the MCCFR framework offers flexibility of tailoring the focus of the algorithm while still maintaining its theoretical guarantees (with high probability) in the long-term. External sampling MCCFR is currently the preferred CFR variant of choice used by the

Computer Poker Research Group for 3-player limit Poker and no-limit Poker, where it has advanced the state of the art, even in the Poker setting [55]. Hyperborean, the winner of the 3-player Heads Up Limit competition, as well as the Bankroll Instant Run-off part of the no-limit competition, is based on external sampling MCCFR [84]. Also, the winning agent of the Heads-Up Limit Poker Competition 2012 was based on an implementation of Public Chance-Sampling (PCS), another MCCFR algorithm described below [84; 50].

4.8 Applications and Extensions

This section contains a survey of the literature describing algorithms which are based on or have extended MCCFR. Each description is a summary of the work[3] and is intentionally brief; for more detail, please see the citations.

4.8.1 Monte Carlo Restricted Nash Responses

The restricted Nash response (RNR) technique is an algorithm for producing robust best responses [54]. Suppose there exists a game G and some fixed popular strategy σ_{fix} that players often employ. Generally playing a best response to this fixed strategy is not advisable because the best response itself can be highly exploitable. The RNR technique describes a new game G' with a single chance node at the top determining, with some probability p, whether the opponent will be restricted to playing σ_{fix}. This new game G' is constructed in such a way that the unrestricted player does not know which subgame they are in (the one with a fixed opponent versus the one with a learning opponent). Running CFR in G' will yield an equilibrium strategy for both players; however, the equilibrium strategy for the unrestricted player will have been constructed from plays against a regret-matching learner $100(1-p)$ % of the time and against the fixed strategy $100p$ % of the time. In their original work, Johanson et. al. (2008) showed that using the equilibrium strategy from G' in G provides the best trade-off (given $p \in [0, 1]$) between exploiting the fixed strategy and being exploitable to an arbitrary rational opponent.

The sampling version of RNR, called MCRNR, uses outcome sampling on this modified game [88]. In that work, MCRNR was shown to converge faster than vanilla RNR in four different domains.

[3]The author of this thesis is a co-author of each of the four publications described in this section.

4.8.2 Public Chance Sampling

Public Chance Sampling (PCS) is another sampling scheme that involves classifying chance nodes into categories: known to all players (public), and known to only the one player (private) [52]. In chance sampling CFR, a single outcome is sampled every time a chance node is encountered; the subtree considered at every iteration is one consistent with all the sampled outcomes. In PCS, only the public chance nodes are sampled (*e.g.,* flop cards in Poker). Private chance nodes can only affect the strategy used by the player who knows the outcome, since these private outcomes determine which information set that player will reach. By reformulating the algorithm sightly, PCS maintains vectors (of size n) of all the players' reach probabilities, one per outcome (of n possible outcomes) of the private chance events, and only the public tree of the game need be traversed and most computations become vector operations.

For example, in Bluff, every chance event is private, so PCS traverses only the public tree defined by all the possible valid bidding sequences rather than the full game tree which also contains chance nodes determining the private rolls of both players. For each player, PCS maintains a vector of reach probabilities, one per private outcome (a roll in Bluff). At leaf nodes these two vectors describe a matrix of possible outcomes given the sequence of public actions (bidding sequence), with each cell corresponding to different possible game outcome due to chance. The expected utility computation for this sequence of public actions is then $O(n^2)$, but can often be reduced by exploiting (domain-dependent) similarities in the utility function over the range of possible outcomes into an $O(n)$ calculation.

4.8.3 Generalized MCCFR and Probing

When using MCCFR, sampling introduces variance in its estimates. In MCCFR algorithms, the sampled counterfactual value for an action that was not sampled in the block of histories is defined to be zero. Generalized MCCFR proposes *any* general estimator of the counterfactual value, $\hat{v}$, and bounds are derived in terms of the variance and bias on the estimates [33].

A new sampling technique, called *probing*, is proposed to reduce the variance of the estimates computed by the MCCFR estimate $\tilde{v}$. When an action a is not sampled at information set I by MCCFR, instead of assuming $\tilde{v}(\sigma_{I \to a}, I) = 0$, an estimate is obtained via a single roll-out of the game to a leaf (a "probe") where both players sample actions on-policy. Probing is shown to reduce the variance and lead to faster convergence in practice in Goofspiel(7), Bluff(2,2), and Texas Hold'em.

4.8.4 Average Strategy Sampling

Since the average strategy $\bar{\sigma}^T$ is the one converging to an equilibrium, it seems sensible that the strategies used during the trials be influenced (informed) by their converging equilibrium strategies; this is precisely what is done in Average Strategy Sampling[34]. For example, if action a and information set I has 0 probability in every equilibrium strategy, the average strategy may have very low probability of taking this action; this information is useful in directing the strategy selection.

Recall from Algorithms 1 to 5 that the average strategy probability at for $a \in A(I)$ at some I is $\bar{\sigma}^T(I,a) = s_I[a] / \sum_{b \in A(I)} s_I[b]$. In average strategy sampling, the subtree below action $a \in A(I)$ is sampled independently with probability

$$\rho(I,a) = \max \left\{ \epsilon, \frac{\beta + \tau s_I[a]}{\beta + \sum_{b \in A(I)} s_I[a]} \right\}$$

or with probability 1 if $\rho(I,a) > 1$ or if the denominator is equal to 0. Here, $\epsilon \in (0,1], \tau \in [1,\infty)$, and $\beta \in [0,\infty)$ are parameters that affect the influence of the average strategy. When $\beta = 0$, τ acts as a threshold: a always gets sampled if its probability is at least $1/\tau$. Early in the trial sequence, the average strategy is not near convergence and so β acts as way to control how much influence there is in the first iterations.

4.9 Chapter Summary and Conclusion

In this chapter, we presented Monte Carlo Counterfactual Regret Minimization, a general family of sample-based algorithms for minimizing counterfactual regret and computing approximate equilibria. The convergence rate of MCCFR depends on the sampling scheme chosen and the structure of the game it is run on. In theory, external sampling has an asymptotically better bound than outcome sampling due to the lack of the $\frac{1}{\delta}$ term. If computation time is the measure of performance, and we fix a tolerance probability p, then the theory claims that external sampling will provide a better convergence rate than Vanilla CFR and chance-sampled CFR. We have shown that the performance of MCCFR in practice is highly game-dependent, but that external sampling MCCFR does tend to have a faster convergence rate than Vanilla CFR and chance-sampled CFR on most of our game domains.

Chapter 5

Regret Minimization in Games with Imperfect Recall

In a game with perfect recall, every player remembers (perfectly) all of the information that was revealed to them and the order in which it was revealed. Most algorithms that compute an equilibrium or an approximate equilibrium for imperfect information games (*e.g.,* Sequence-form Linear Programming, Counterfactual Regret Minimization) require that games have this property of perfect recall.

From a computational standpoint, the necessity of perfect recall is unfortunate since $|\mathcal{I}_i|$ grows quickly with every bit of information revealed to each player i. In some cases, requiring perfect recall is outright wasteful. As an extreme example, suppose two players choose to play a regular game of Texas Hold'em poker. Before the game, they flip a coin; the outcome of the coin flip does not affect the game whatsoever. This is a new game with twice as many information sets. As a result, algorithms consume twice as much memory and can take twice as long to compute the same ϵ-equilibrium.

In single-agent settings, the Markov property allows an algorithm to forget information from the past states, allowing the algorithm to make decisions based only on the current state. No such property exists in the multi-player imperfect information setting. However, clearly there are instances (as in the example above) where forgetting a piece of information *should* be safe and have no effect on the strategies players would use to play the game.

In recent years, members of the Computer Poker Research Group (CPRG) have applying chance-sampled CFR to abstract poker games. Suppose we have a game (*e.g.,* Two-Player Limit Texas Hold'em). An abstract game is generally a smaller game; we will consider abstractions that are obtained by merging information sets together. The aim is to merge "similar" or identical information in the sense that any information lost in the merge should be strategically unimportant. Then, the process is to find an equilibrium in the small

game and convert the strategy to one usable in the larger game. Some of the CPRG abstractions are perfect recall abstractions (*i.e.,* the resulting smaller game also has perfect recall). For example, Poker hands are classified into *buckets*, which are classifications of chance node outcomes into a coarser set as a function of some notion of hand strength [51, Section 2.5]. Over the last few years, the CPRG have also used a form of abstraction based on forgetting the bucket their hand was classified into from previous rounds, which can dramatically decrease the overall size of the game and performs well in practice [118]. However, CFR is not guaranteed to compute an approximate equilibrium in an imperfect recall game. In this chapter, we will show that in fact convergence can be guaranteed under certain conditions.

5.1 Imperfect Recall Games

Recall from Section 2.1.2 that a game has perfect recall if

$$\forall i \in N, \forall I \in \mathcal{I}_i, \forall h, h' \in I : X_i(h) = X_i(h'),$$

where $X_i(h)$ is the sequence of (information set, action) pairs taken by player i in the same order that they are taken in h.

In a game of perfect recall, what a player knows at I and knew at every step up to I must be consistent for every history part of I. What this definition effectively guarantees is that player i remembers any information revealed to them during the game and the order in which it was revealed.

Perfect recall is very commonly assumed in extensive-form games with imperfect information because otherwise fundamental results no longer hold. The most well-known example of such problems is shown through the Absent-Minded Driver paradox [83]. Consider the single-player game in Figure 5.1a. The problem is this: a man has to drive home from a bar. There are two intersections along the way, where he can choose to turn right or to keep going straight. If he turns right at either intersection, the game ends immediately. The man is absent-minded – and the intersections are strikingly similar looking! – so he doesn't remember if he has been through an intersection or not when arriving at one. Turning right at the first intersection means getting lost and results in a payoff of 0. If he turns right at the second intersection, he makes it home and receives a payoff of 4. Going straight twice results in a payoff of 1, resulting in a longer way home. Piccione and Rubinstein claimed that this creates a paradox; before leaving the bar the driver considers the pure strategy of driving straight to be optimal, since turning right will lead to 0. But

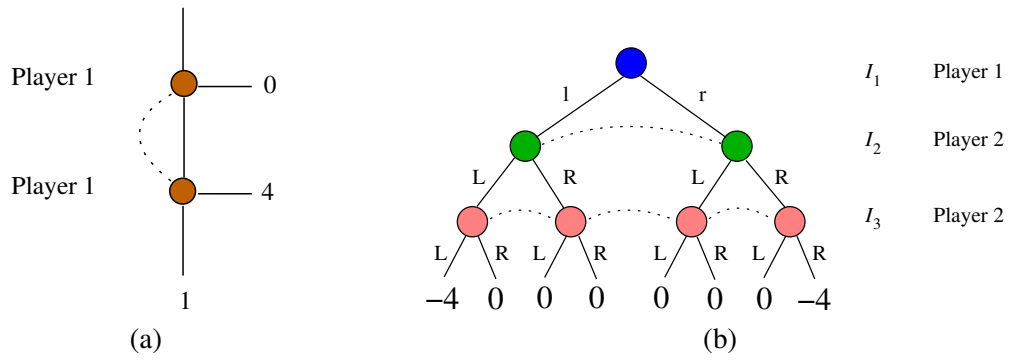


Figure 5.1: Two examples of games with imperfect recall.

when coming up to any given intersection, it is the correct one with probability $\frac{1}{2}$, and so there is incentive to turn right. This means that the driver's rationality is inconsistent: if no information changes and players' utilities remain constant, there should be no difference in reasoning before and during the game. If the driver is allowed to randomize, then a similar argument can be constructed; the optimal strategy apriori is for the driver to turn right with probability $\frac{1}{3}$ giving expected payoff $\frac{4}{3}$. Yet, when the intersection is reached, it is the correct one with probability $\frac{2}{5}$, so the value of turning is $\frac{2}{5} \cdot 4 + \frac{3}{5} \cdot 0 = \frac{8}{5}$. Again, the driver would rather always turn, which is inconsistent with the initial plan.

Now suppose the payoff for going straight were changed to 0, then there would be two pure strategies that give payoffs of 0. Any mixed strategy over these two in a perfect recall game will always result in an expected payoff of 0, but a mixed strategy of $(\frac{1}{2}, \frac{1}{2})$ leads to an expected payoff of 0.25 in this game. In effect, mixed and behavioral strategies are no longer equivalent in imperfect recall games. This difference between perfect and imperfect recall games was first observed by Kuhn in his initial work on extensive form games [66].

As another example, consider the game shown in Figure 5.1b, based on [62, Example 2.4]. Here, there are no chance events and only player 2 has imperfect recall: after taking their first action, they immediately forget which one they took. The pure strategy sets are $S_1 = \{l, r\}$ and $S_2 = \{LL, LR, RL, RR\}$. The optimal mixed strategy for player 1 is to choose l and r each with probability $\frac{1}{2}$. Similarly, player 2 also mixes equally between LL and RR, assigning 0 to both LR and RL. If players could employ these mixed strategies, player 2 would assure themselves an expected payoff of 2. However, if player 2 employs a behavioral strategy with $\sigma(I_2, \mathrm{L}) = x$ and $\sigma(I_3, \mathrm{L}) = y$ then the payoff is at most

$$\max_{x,y \in [0,1]} \min\{4xy, 4(1-x)(1-y)\} = 1,$$

which is no longer optimal for player 2. Therefore, the mixing over pure strategies allows the player to make use of information that is not allowed by the structure of the game and results in a higher payoff.

In this chapter we relax the requirement of perfect recall for CFR-based algorithms by deriving regret bounds for a general class of imperfect recall games. We present two classes of imperfect recall games that are motivated by the use of abstraction. The first class is composed of imperfect recall games were information that is truly irrelevant is shown to be safe to forget. The second quantifies a regret penalty that is paid when one of the conditions required by the first class is relaxed. In addition, we present empirical results to analyze the effects of games from both of these classes as well as further evidence or practical uses of abstractions outside these classes, adding to previous results in Poker [118]. The contents of this chapter form an expanded version of the recently published paper at the International Conference on Machine Learning [67]. In particular, Richard Gibson, Neil Burch, and Michael Bowling contributed to the development of the content in this chapter, which was initially based on previous unpublished work by Martin Zinkevich, Kevin Waugh, and Michael Bowling.

We concentrate on adapting the original CFR algorithm to handle imperfect recall games. Therefore, the algorithm we use to apply our ideas is the original CFR algorithm, not Monte Carlo CFR.

5.2 Well-Formed Imperfect Recall Games

We start with some definitions that will be needed throughout the chapter. Define $X_{-i}(h)$ to be the (information set, action) pairs belonging to i's opponent in the same order that they were encountered and taken in h. Define $X(h)$ to be the full interleaving sequence of (information set, action) pairs encountered and taken by both players along h. Define $X(h, z)$ to be the sequence of (information set, action) pairs from h to z, where h is a prefix of z, and define $X_i(h, z)$ and $X_{-i}(h, z)$ similarly as above.

Throughout the chapter, we will use a running example. Recall the game Die-Roll Poker (DRP(N)) from Section 3.1.3. DRP is naturally a game with perfect recall; players remember the exact sequence of bets made and the exact outcome of each die roll from both rounds. Consider now the imperfect recall version of DRP, called DRP-IR, where at the beginning of the second round, both players forget their first die roll and only know the sum of their two dice. DRP-IR is an abstraction of DRP where any two histories are in the

same (abstract) information set if and only if the sum of the player's private dice is the same and the sequence of betting is the same. DRP-IR has imperfect recall since histories that were distinguishable in the first round (*e.g.,* a ⚀ and a ⚁) are no longer distinguishable in the second round (*e.g.,* a ⚀ followed by a ⚅ and a ⚁ followed by a ⚄, since both sum to 6).

We begin by restricting our discussion to games where players cannot reach the same information set twice in a single game. We say that a game is **strictly informative** if for all $i \in N$ and $h, h' \in H_i$

$$h \sqsubseteq h', h \neq h' \Rightarrow I(h) \neq I(h'), \tag{5.1}$$

where $I(h)$ represents the information set containing h. For example, the absent-minded driver is not strictly informative but the game shown in Figure 5.1(b) is. In this thesis, we only consider games that are strictly informative; this is because the counterfactual value, Equation 2.15, would not be well-defined otherwise (since the prefix $z[I]$ could not be unique).

We will say that a game $\Gamma' = \langle N, A', H, Z, P, \sigma_c, u, \mathcal{I}' \rangle$ is an **abstract game**, or **abstraction**, of $\Gamma = \langle N, A, H, Z, P, \sigma_c, u, \mathcal{I} \rangle$ if for all $i \in N$ and $h, k \in H_i$: $I(h) = I(k)$ implies $I'(h) = I'(k)$, where $H_i = \{h \mid h \in H, P(h) = i\}$. A typical use of abstraction is to reduce the size of the game by ensuring that $|\mathcal{I}'| < |\mathcal{I}|$.

For games $\Gamma = \langle N, A, H, Z, P, \sigma_c, u, \mathcal{I} \rangle$ and $\breve{\Gamma} = \langle N, A, H, Z, P, \sigma_c, u, \breve{\mathcal{I}} \rangle$, we say that $\breve{\Gamma}$ is a **perfect recall refinement of Γ** if $\breve{\Gamma}$ has perfect recall and Γ is an abstraction of $\breve{\Gamma}$. The information available to players in $\breve{\Gamma}$ is never forgotten, and is at least as informative as the information available to them in Γ. For example, DRP is a perfect recall refinement of DRP-IR. Every game has at least one perfect recall refinement by simply making $\breve{\Gamma}$ a perfect information game ($\breve{I} = \{h\}$ for all $\breve{I} \in \breve{\mathcal{I}}_i$). Furthermore, a perfect recall game is a perfect recall refinement of itself. For $I \in \mathcal{I}_i$, we define

$$\breve{\mathcal{P}}(I) = \{\breve{I} \mid \breve{I} \in \breve{\mathcal{I}}_i, \breve{I} \subseteq I\}$$

to be the set of all information sets in $\breve{\mathcal{I}}_i$ that are subsets of I. Note that our notion of refinement is similar to the one described by Kaneko & Kline in [56]. Our definition differs in that we consider any possible refinement, whereas Kaneko & Kline consider only the coarsest such refinement.

Definition 1. *For a game Γ and a perfect recall refinement $\breve{\Gamma}$, we say that Γ is a **well-formed game with respect to $\breve{\Gamma}$** if for all $i \in N$, $I \in \mathcal{I}_i$, $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$, there exists a bijection $\phi : Z_{\breve{I}} \to Z_{\breve{I}'}$ and constants $k_{\breve{I},\breve{I}'}, \ell_{\breve{I},\breve{I}'} \in [0, \infty)$ such that for all $z \in Z_{\breve{I}}$:*

(i) $u_i(z) = k_{\breve{I},\breve{I}'} u_i(\phi(z))$,

(ii) $\pi_c(z) = \ell_{\breve{I},\breve{I}'} \pi_c(\phi(z))$,

(iii) *In* Γ, $X_{-i}(z) = X_{-i}(\phi(z))$, *and*

(iv) *In* Γ, $X_i(z[\breve{I}], z) = X_i(\phi(z)[\breve{I}'], \phi(z))$.

We say that Γ *is a* ***well-formed game*** *if it is well-formed with respect to some perfect recall refinement.*

Recall that Z_I is the set of terminal histories containing a prefix in the information set I, and that $z[I]$ is that prefix. Intuitively, a game is well-formed if for each information set $I \in \mathcal{I}_i$, the structures around each $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$ of some perfect recall refinement are isomorphic across four conditions. Conditions (i) and (ii) state that the corresponding utilities and chance frequencies at each terminal history are proportional. Condition (iii) asserts that the opponents can never distinguish the corresponding histories at any point in Γ. Finally, condition (iv) states that player i cannot distinguish between corresponding histories from $\breve{I}$ and $\breve{I}'$ until the end of the game.

Consider again DRP as a perfect recall refinement of DRP-IR. In DRP, the available actions are independent of dice outcomes, and the final utilities are only dependent on the final sum of the players' dice. Therefore, in DRP the utilities are equivalent between, for example, the terminal histories where player i rolled a ⚀ followed by a ⚄, and the terminal histories where player i rolled a ⚃ followed by a ⚁ (condition (i)). In addition, the chance probabilities of reaching each terminal history are equal (condition (ii)). Furthermore, the opponents can never distinguish between two isomorphic histories since player i's rolls are private (condition (iii)). Finally, in DRP-IR, player i never remembers the outcome of the first roll from the second round on (condition (iv)). Thus, DRP-IR is well-formed with respect to DRP, with constants $k_{\breve{I},\breve{I}'} = \ell_{\breve{I},\breve{I}'} = 1$.

Any perfect recall game is well-formed with respect to itself since $\breve{\mathcal{P}}(I) = \{I\}$, ϕ equal to the identity bijection, and $k_{\breve{I},\breve{I}'} = \ell_{\breve{I},\breve{I}'} = 1$ satisfies Definition 1. However, many imperfect recall games are also well-formed, with DRP-IR being one example. An additional example is presented in Section 5.5. We now show that CFR can be applied to any well-formed game to minimize average regret.

Theorem 6. *If* Γ *is well-formed with respect to* $\breve{\Gamma}$*, then the average regret in* $\breve{\Gamma}$ *for player* i *of choosing strategies according to CFR in* Γ *is bounded by*

$$\frac{\breve{R}_i^T}{T} \leq \frac{\Delta_i K \sqrt{|A_i|}}{\sqrt{T}},$$

where $K = \sum_{I \in \mathcal{I}_i} \max_{\breve{I},\breve{I}' \in \breve{\mathcal{P}}(I)} k_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'}$.

The proof of this theorem is presented in Appendix A.2. The result of this theorem is that counterfactual regret minimization can be applied in the imperfect recall game. For example, in DRP-IR, since $k_{\breve{I},\breve{I}'} = \ell_{\breve{I},\breve{I}'} = 1$ the regret bound works out to be equivalent to the CFR regret bound in [124], except that $|\mathcal{I}|$ in the abstract game is much smaller. There is a subtle but important point about this result regarding abstraction: for some game Γ and a *perfect recall abstraction* of Γ, say Γ': Γ' is not necessarily always well-formed by Definition 1. However, it is possible to have two games with perfect recall, one being a well-formed abstraction of the other, where this result will hold; one example is a game and its equilibrium-preserving lossless abstraction obtained by GameShrink [35, Chapter 5]. In general, abstraction (even if both games have perfect recall) may create pathological situations that pose a problem for regret minimization and equilibrium computation [117].

How can the CFR algorithm be applied with these new definitions? In fact, it is quite straight-forward; the algorithm we discuss in this chapter is almost identical to Algorithm 1 from Chapter 2. There are two notable changes. The first is that the information set on line 13 is within the imperfect recall game Γ, not its perfect recall refinement $\breve{\Gamma}$. The second is that for proper averaging, the average strategy is maintained in the full game. In particular, h is always a valid sequence of actions in the full game $\breve{\Gamma}$. The external regret that is being minimized is the one in the perfect recall refinement. The reduction of the state space allows CFR to be applied to a smaller game. However, if computation of an ϵ-equilibrium is the ultimate goal, then the computation of the average strategy is problematic, since it must be done in the perfect recall refinement. We discuss this issue is in detail in Section 5.4.

5.3 Skew Well-Formed Imperfect Recall Games

We now present a generalization of well-formed games for which a regret bound can still be derived.

Definition 2. *For a game* Γ *and a perfect recall refinement* $\breve{\Gamma}$*, we say that* Γ *is a* ***skew well-formed game with respect to*** $\breve{\mathbf{\Gamma}}$ *if for all* $i \in N$*,* $I \in \mathcal{I}_i$*,* $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$*, there exists a bijection* $\phi : Z_{\breve{I}} \to Z_{\breve{I}'}$ *and constants* $k_{\breve{I},\breve{I}'}, \delta_{\breve{I},\breve{I}'}, \ell_{\breve{I},\breve{I}'} \in [0, \infty)$ *such that for all* $z \in Z_{\breve{I}}$*:*

(i) $\left| u_i(z) - k_{\breve{I},\breve{I}'} u_i(\phi(z)) \right| \leq \delta_{\breve{I},\breve{I}'}$,

(ii) $\pi_c(z) = \ell_{\breve{I},\breve{I}'} \pi_c(\phi(z))$,

(iii) *In* Γ, $X_{-i}(z) = X_{-i}(\phi(z))$, *and*

(iv) *In* Γ, $X_i(z[\breve{I}], z) = X_i(\phi(z)[\breve{I}'], \phi(z))$.

We say that Γ *is a* ***skew well-formed game*** *if it is skew well-formed with respect to some perfect recall refinement.*

The only difference between Definitions 1 and 2 is in condition (i). While utilities must be exactly proportional in a well-formed game, in a skew well-formed game they must only be proportional up to a constant $\delta_{\breve{I},\breve{I}'}$. Note that any well-formed game is skew well-formed by setting $\delta_{\breve{I},\breve{I}'} = 0$.

For example, consider a new version of DRP called Skew-DRP(δ) with slightly modified payouts at the end of the game. Whenever the game reaches a showdown, player 1 receives a bonus δ times the number of chips in the pot from player 2 if player 1's second die roll was even; otherwise, no bonus is awarded. The pot is then awarded to the player with the highest dice sum as usual. Analogously, define Skew-DRP-IR(δ) to be the imperfect recall abstraction of Skew-DRP(δ) where in the second round, players only remember the sum of their two dice. Now, Skew-DRP-IR(δ) is not well-formed with respect to Skew-DRP(δ). To see this, note that the utilities resulting from the rolls ⚀,⚄ and the rolls ⚃,⚀ and the same sequence of betting are not exactly proportional because the second roll ⚄ is odd but ⚀ is even (utilities are off by δ times the pot size). However, Skew-DRP-IR(δ) is skew well-formed with respect to Skew-DRP(δ) with $\delta_{\breve{I},\breve{I}'} = \delta$ times the maximum pot size attainable from I.

Unfortunately, there is no guarantee that regret will be minimized by CFR in a skew well-formed game. However, we can still bound regret in a predictable manner according to the degree in which the utilities are skewed:

Theorem 7. *If* Γ *is skew well-formed with respect to* $\breve{\Gamma}$*, then the average regret in* $\breve{\Gamma}$ *for player* i *of choosing strategies according to CFR in* Γ *is bounded by*

$$\frac{\breve{R}_i^T}{T} \leq \frac{\Delta_i K \sqrt{|A_i|}}{\sqrt{T}} + \sum_{I \in \mathcal{I}_i} |\breve{\mathcal{P}}(I)| \delta_I,$$

where $K = \sum_{I \in \mathcal{I}_i} \max_{\breve{I},\breve{I}' \in \breve{\mathcal{P}}(I)} k_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'}$ *and* $\delta_I = \max_{\breve{I},\breve{I}' \in \breve{\mathcal{P}}(I)} \delta_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'}$.

The proof is similar to that of Theorem 6 and is included in Section A.2. Theorem 7 shows that as T approaches infinity, the bound on our regret approaches $\sum_{I \in \mathcal{I}_i} |\breve{\mathcal{P}}(I)| \delta_I$. Our experiments in Section 5.5 demonstrate that as the skew δ grows, so does our regret in Skew-DRP(δ) after a fixed number of iterations.

Remarks. Theorems 6 and 7 are, to our knowledge, the first to provide such theoretical guarantees in imperfect recall settings. However, these results are also relevant with regards to regret in the full game when CFR is applied to an abstraction. Recall that if Γ has perfect recall, then Γ is a perfect recall refinement of any (skew) well-formed abstract game. Thus, if we choose an abstraction that yields a (skew) well-formed game, then applying CFR to the abstract game achieves a bound on the average regret *in the full game*, Γ. This is true regardless of whether the abstraction exhibits perfect recall or imperfect recall. Previous counterexamples show that abstraction in general provides no guarantees in the full game [117]. In contrast, our results show that applying CFR to an abstract game leads to bounded regret in the full game, provided we restrict ourselves to (skew) well-formed abstractions. If such an abstract game is much smaller than the full game, a significant amount of memory is saved when running CFR.

5.3.1 Relaxing the Conditions

Well-formed games are described by four conditions provided in Definition 1. Recall that Koller & Megiddo prove that determining a player's guaranteed payoff in an imperfect recall game is NP-complete [62]. However, Koller & Megiddo's NP-hardness reduction creates an imperfect recall game that breaks conditions (i), (iii), and (iv) of Definition 1. In this section, we discuss the following question: For minimizing regret, how important is it to satisfy each individual condition of Definition 1?

Skew well-formed games and Theorem 7 show that one can relax condition (i) of Definition 1 and still derive a bound on the average regret. In addition, most of our PTTT and Bluff abstractions from the Section 5.5 do not satisfy condition (iii), but CFR still produces reliable results. This suggests that it may be possible to relax condition (iii) in a similar manner to the relaxation of condition (i) introduced by skew well-formed games. While we leave this question open, we now demonstrate that breaking condition (iii) can lead CFR to a dead-lock situation where one player has constant average regret.

Let us walk through the process of applying CFR to the game in Figure 5.2. Note that this game satisfies all of the conditions of Definition 1, except for condition (iii). To begin, the current strategy profile σ^1 is set to be uniform random at every information set. Under this profile, when player 1 is at I_3, each of the four histories are equally likely. Thus, $v_i(\sigma^1_{(I_3 \to l)}, I_3) = v_i(\sigma^1_{(I_3 \to r)}, I_3) = v_i(\sigma^1, I_3) = 0$, and so $r^1_1(I_3, l) = r^1_1(I_3, r) = 0$. Similarly for actions p and c at I_1 and I_2. Player 2, however, has positive immediate counterfactual regret for passing (p) at histories ac and ec (to always receive ξ utility) and

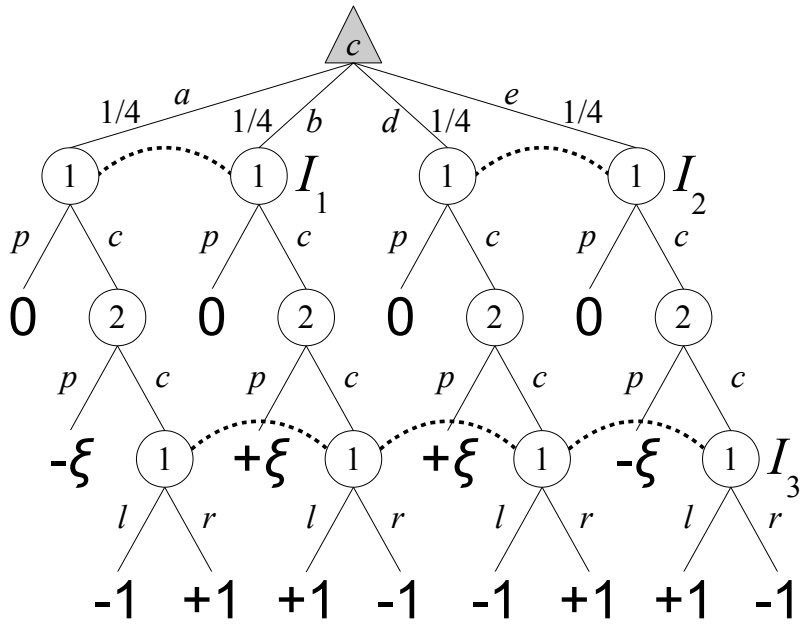


Figure 5.2: A zero-sum game with imperfect recall where CFR does not minimize average regret. The utilities for player 1 are given at the terminal histories, where $\xi \in (0, 1)$. Nodes connected by a bold, dashed curve are in the same information set for player 1 (player 2 has perfect information).

for continuing (c) at bc and de (to always avoid receiving $-\xi$ utility), and has negative immediate counterfactual regret for continuing at ac and ec and for passing at bc and dc. Therefore, the next profile σ^2 still has player 1 playing uniformly random everywhere, but player 2 now always passes at ac and ec, and always continues at bc and dc. On the second iteration of CFR, the positive regrets for player 1 at I_3 remain the same because the histories bcc and dcc are equally likely. Also, player 2's positive regrets remain the same at all four histories in H_2. However, player 1's expected utility for continuing at I_1 or I_2 is now negative since player 2 now passes at ac and ec, and player 1 gains positive regret for passing at both I_1 and I_2. This leads us to the next profile $\sigma^3 = \{(I_1, p) = 1, (I_2, p) = 1, (ac, p) = 1, (bc, p) = 0, (dc, p) = 0, (ec, p) = 1, (I_3, l) = 0.5\}$. One can check that running CFR for more iterations yields $\sigma^t = \sigma^3$ for all $t \geq 3$. The average regret for playing this way will be constant and hence does not approach zero because player 1 would rather play $\sigma'_1 = \{(I_1, p) = 1, (I_2, p) = 0, (I_3, l) = 0\}$ and get $u_1(\sigma'_1, \sigma^3_2) = (1 - \xi)/4 > u_1(\sigma^3)$ for $\xi \in (0, 1)$. A similar example can be constructed where condition (iii) holds, but chance's probabilities are not proportional (breaking condition (ii)).

In some sense, the example highlights the importance of conditions (ii) and (iii): breaking either of these in isolation can lead to a loss of convergence guarantees. If (ii) or (iii)

can be relaxed, then the relaxation must take these tricky scenarios into account. Despite the problem of breaking condition (iii), condition (iv) of Definition 1 can be relaxed. Rather than enforcing player i's future information to be the same across the bijection ϕ, we only require that the corresponding subtrees be isomorphic, allowing player i to re-remember information that was previously forgotten. The details for this relaxation are presented in the extended technical report [68]. It is not clear that this relaxation is possible in skew well-formed games, nor does it seem to provide any practical advantage, so we omit its presentation in this thesis.

5.4 Average Strategy Computation

Recall the folk theorem property from Section 2.2.2: when running the CFR algorithm, the strategy that is converging to an equilibrium is the average strategy (Equation 2.14). What we have shown above is that minimizing regret in a well-formed imperfect recall game Γ will also minimize regret in its perfect recall refinement $\breve{\Gamma}$. However, the strategy that is converging is the average strategy in $\breve{\Gamma}$. In general, for $I \in \mathcal{I}$,

$$\bar{\sigma_i}^T(I,a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I,a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)} = \frac{\sum_{t=1}^{T} \sum_{h \in I} \pi_i^{\sigma^t}(h)\sigma^t(I,a)}{\sum_{t=1}^{T} \sum_{h \in I} \pi_i^{\sigma^t}(h)} \tag{5.2}$$

will not be equal to $\bar{\sigma}(\breve{I}, a)$ for $\breve{I} \in \breve{\mathcal{P}}(I)$.

To see why this is true, imagine two separate prefix histories of DRP-IR:

h_1 : Player i rolls ⚀, $-i$ rolls something, check, raise, call, i rolls ⚅, $-i$ rolls something.

h_2 : Player i rolls ⚅, $-i$ rolls something, check, raise, call, i rolls ⚀, $-i$ rolls something.

At this point it is player i's turn. In DRP-IR, these two prefix histories are merged into one information set I. However, previously (before player i checks), these histories are were in two different information sets. Hence, the product $\pi_i^{\sigma}(h_1) \neq \pi_i^{\sigma}(h_2)$ and the sum in the numerator of Equation 5.2 will add these together since $h_1, h_2 \in I$.

One solution is to maintain and update the converging average strategy $\{\bar{\sigma}(\breve{I}) : \breve{I} \in \breve{\mathcal{I}}\}$. This is possible using the CFR algorithm since the only difference is in how the information sets are retrieved given history h (Line 13 or Algorithm 1). This requires maintaining two separate data structures, one for the abstract game Γ and one for full game $\breve{\Gamma}$. However, if the abstraction is used to save memory, this is undesirable as it still requires memory $O(|\breve{\mathcal{I}}|)$ which is what is being avoided. We refer to this averaging scheme as **full averaging**.

So, as with MCCFR, we can resort to heuristic approximations. One reasonable approximation is to simply save the strategy profiles at random iterations $T_{\text{save}} = \{t_1, t_2, \ldots\} \subset \{1, 2, \cdots, T\}$ to obtain a collection of profiles $(\sigma^{t_1}, \sigma^{t_2}, \sigma^{t_3}, \cdots)$. Then, when employing the strategy at information set I, use these strategies to compute a mixed $\bar{\sigma}(\breve{I})$ on-the-fly from the saved strategies. This will require an amount of memory that is linear in the size of the abstract game multiplied by $|T_{\text{save}}|$.

Another approximation is to simply ignore the problem altogether. Computing $\bar{\sigma}$ in the abstract game effectively combines all the average strategy updates from $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$. As a result, the strategy recommended by $\bar{\sigma}$ will be a combination of what was learned along the different prefixes in I. Since these prefixes are being combined to form I in the first place, this seems sensible. We refer to this averaging scheme as **abstract averaging**.

In the next section, we will evaluate the performance of this in practice versus the proper way to compute the average.

5.5 Empirical Evaluation

To complement the theoretical results, we apply CFR to both players simultaneously in several zero-sum imperfect recall (abstract) games, and measure the sum of the average regrets for both players in a perfect recall refinement (the full game). Along with the small DRP domain and its variants, we also consider the challenging domains of phantom tic-tac-toe and Bluff, described in sections 3.1.1 and 3.1.7.

We consider several different imperfect recall abstractions for DRP, Skew-DRP(δ), PTTT, and Bluff. For the DRP games, we apply DRP-IR and Skew-DRP-IR(δ) respectively as described in sections 5.2 and 5.3.

In Bluff, we use abstractions described by Neller and Hnath ([77]) that force players to forget everything except the last r bids. These abstract games are not skew well-formed because the players forget information that the opponent could previously distinguish.

In our first set of experiments, we evaluate the performance of full averaging versus abstract averaging. We run two experiments using Vanilla CFR, on two separate games: DRP-IR, and Bluff(1,1) with $r = 4$. We apply the full averaging mechanism as well as the abstract averaging mechanism described in Section 5.4. The results, shown in Figure 5.3, are encouraging. In both cases, the difference between the two is hardly noticeable. In the case of DRP-IR, the values are in fact identical while in the case of Bluff(1,1) there are small differences in the fourth decimal place. This shows promise for the abstract averaging

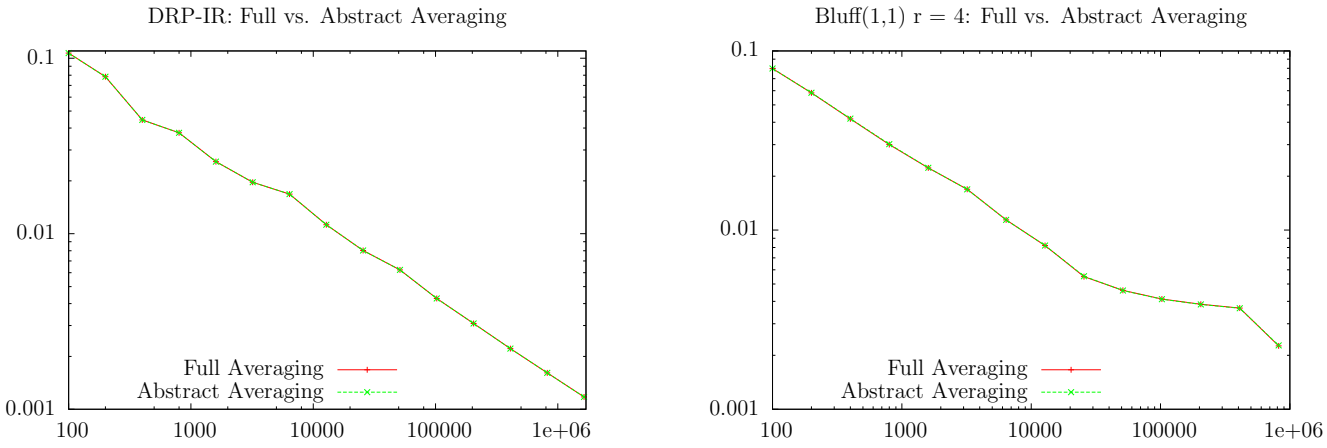


Figure 5.3: Full averaging versus abstract averaging in DRP-IR and Bluff(1,1) $r = 4$. The horizontal axes represent iterations while the vertical axes represents exploitability ϵ_σ in the full game.

approach. However, more experiments and formal analysis are required to say anything further. We include these results mainly to encourage further research; the remainder of our experiments use full averaging.

The next set of experiments compares the performance of the algorithm applied to much larger games. Our PTTT and Bluff experiments also investigate the effects of imperfect recall beyond skew well-formed games. In the full, perfect recall version of PTTT, each player remembers the order of every failed and every successful move she makes throughout the entire game. In our first abstract game, **FOSF**, players forget the order of successive failures within the same turn. Clearly, there is an isomorphism between any two merged information sets $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$ since the order of the actions does not affect the available future moves or utilities. Players still remember which turn each success and each failure occurred, and so the opponent's sequences of actions must be equal across the isomorphism. Thus, FOSF is well-formed. The remaining PTTT abstractions, however, are not even skew well-formed. In **FOI**, players independently remember the sequence of failures and the sequence of successful actions, but not how the actions interleave. In **FOS**, players remember the order of failed actions, but not the order of successes. Finally, in **FOE**, players only know what actions they have taken and remember nothing about the order in which they were taken. FOI, FOS, and FOE are not skew well-formed because no isomorphism can preserve the order of the opponent's previous (I, a) pairs (breaking condition (iii) of Definitions 1 and 2).

Game	Abstr.	Well-for.	$\lvert\mathcal{C}\rvert$	Savings
DRP	None	Yes	2610	—
DRP	DRP-IR	Yes	860	67.05%
PTTT	None	Yes	11695314	—
PTTT	FOSF	Yes	9347010	20.08%
PTTT	FOI	No	1147530	90.19%
PTTT	FOS	No	1484168	87.31%
PTTT	FOE	No	47818	99.59%
Bluff	None	Yes	704643030	—
Bluff	$r = 10$	No	295534218	58.06%
Bluff	$r = 8$	No	108323418	84.63%
Bluff	$r = 6$	No	22518468	96.80%
Bluff	$r = 4$	No	2329068	99.67%
Bluff	$r = 3$	No	543900	99.92%
Bluff	$r = 2$	No	97608	99.97%
Bluff	$r = 1$	No	12600	99.99%

Table 5.1: DRP, PTTT, and Bluff game sizes and properties. Here, $|\mathcal{C}|$ represents the total number of (information set, action) pairs for both players.

The size of each DRP, PTTT, and Bluff game is given in Table 5.1, where we define $|\mathcal{C}| = |\mathcal{C}_1 \cup \mathcal{C}_2|$ to be the total number of (I, a) pairs for both players. Note that Skew-DRP(δ) is the same size as DRP regardless of the skew, and recall that CFR requires space linear in $|\mathcal{C}|$.

For each game, we ran CFR on both players, meaning that each player's opponent was an identical copy of the same no-regret learner. Similar to Zinkevich *et al.* ([124]), we used the chance sampling variant of CFR. The sum of the average positive regrets for each player over a number of iterations is shown in Figure 5.4. The Skew-DRP-IR(δ) experiments show that as δ increases, so does the regret as predicted by Theorem 7, though $\sum_{I \in \mathcal{I}_i} \left|\breve{\mathcal{P}}(I)\right| \delta_I$ appears to be a very loose bound on the final regret.

In PTTT, regret appears to diverge from zero for FOI, FOS, and FOE, where FOS appears to provide slightly better strategies than FOI and FOE. While our theory cannot explain why FOS performs better, this does match our intuition that remembering information about the opponent's moves is important, and the importance of conditions (iii) demonstrated in Section 5.3.1. For a small increase in average regret, FOS reduces the space required by 87% compared to FOSF's 20% reduction. Note that for both DRP and PTTT, running CFR on the full, perfect recall game achieves the same regret as in the well-formed abstractions (Skew-DRP-IR(0) and FOSF) and is thus not shown. In Bluff, we see that regret consistently worsens as fewer previous bids are remembered. This suggests that a result

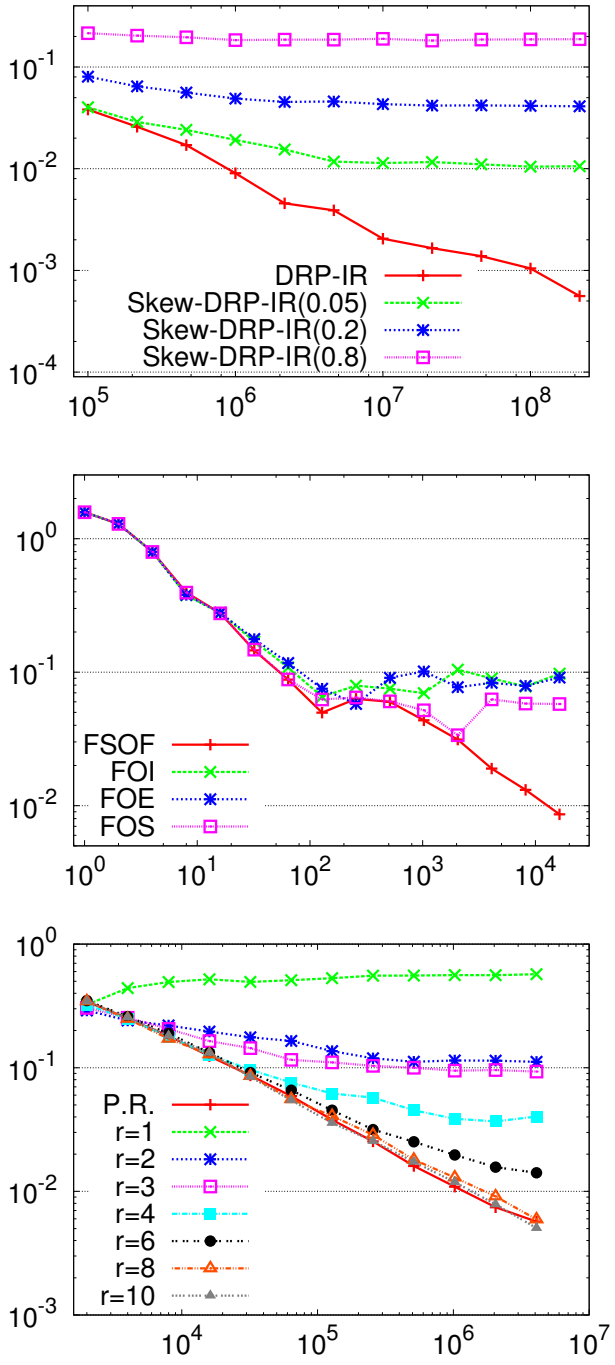


Figure 5.4: Sum of average regrets for both players, $(R_1^{T,+} + R_2^{T,+})/T$, as iterations increase for Skew-DRP-IR(δ) (top), abstract games in PTTT (middle), and abstract games in Bluff(2,2) (bottom). Each graph uses a log scale on both axes. The vertical axes represent the sum of average regret for both players in the corresponding full, unabstracted game, and horizontal axes represent iterations.

similar to Theorem 2 for skew well-formed games may hold if condition (iii) of Definition 2 is less constrained, though the proper formulation for such a relaxation remains unclear. Nonetheless, choosing $r = 8$ saves 85% of the memory with only a very small increase in average regret after millions of iterations.

5.6 Chapter Summary and Conclusion

In this chapter, we have given a definition of well-formed game and skew well-formed games for games with imperfect recall. We have shown that running CFR on well-formed abstract games will still minimize regret in the full game and hence the full average strategy approaches an ϵ-equilibrium. We have also given a relaxed definition that allows for skewing of payoff, and derived the regret penalty that is paid for such a skew. We have also shown the empirical performance of CFR run on well-formed games and skew well-formed games, as well as games that are neither. To save memory, the average abstract strategy must be accumulated, and while this appears to work well in practice, more formal analysis is required to show how well this average abstract strategy approximates the full average strategy.

Chapter 6

Monte Carlo *-Minimax Search

The previous two chapters described algorithms for generating approximate equilibria in large imperfect information games. In games of imperfect information, computing a *mixed* Nash equilibrium may be important; players may be required to mix their strategies to make it difficult for their opponent to determine their private information.

In this and the next chapter, we focus on game-playing algorithms for perfect information games. As such, we will use the terminology associated with these models and their literature as presented in Section 2.3.2.

Recall Monte-Carlo Tree Search (MCTS) from Section 2.3.2. MCTS has recently become one of the dominant paradigms for online planning in large sequential games. At first, MCTS was applied to games lacking strong Minimax players, but recently has been shown to compete against strong Minimax players [90; 120]. MCTS performs particularly well when a good evaluation function is unavailable and the branching factor is high. This is a bad case for Minimax because of the exponential running time in the look-ahead depth. In MCTS, the simulation always continue to the end of the game, so the returns are based on true utility values.

Unlike classical games such as Chess and Go, stochastic game trees include chance nodes in addition to decision nodes. How MCTS should account for this added uncertainty remains unclear. The classical algorithms for stochastic games, Expectimax and *-Minimax, perform look-ahead searches to a limited depth. However, both algorithms scale exponentially in the branching factor at chance nodes as the search horizon is increased. Hence, their performance in large games often depends heavily on the quality of the heuristic evaluation function, as only shallow searches are possible.

One way to handle the uncertainty at chance nodes is to simply sample a single outcome when encountering a chance node. This is common practice in MCTS when applied

to stochastic games; however, the general performance of this method is unclear. Large stochastic domains still pose a significant challenge. For example, MCTS is outperformed by *-Minimax in the game of Carcassonne [45]. Unfortunately, the literature on the application of Monte Carlo search methods to stochastic games is relatively small, possibly due to the lack of a principled and practical algorithm for these domains.

In this chapter, we introduce a new algorithm, called Monte-Carlo Minimax Search (MCMS), which can increase the performance of search algorithms in stochastic games by sampling a subset of chance event outcomes. The work presented in this chapter is an extended version of our work recently presented at the Computer Games Workshop held at the European Conference on Artificial Intelligence [69], done in collaboration with Abdallah Saffidine, Joel Veness, and Chris Archibald.

6.1 Ballard's *-Minimax

Recall classical game search from Section 2.3.2. A direct computation of

$$\arg\max_{a \in \mathcal{A}(s)} V_d(s,a) \qquad \text{or} \qquad \arg\min_{a \in \mathcal{A}(s)} V_d(s,a)$$

(see Equation 2.20) in the stochastic setting is known as the Expectimax algorithm [94]. The base Expectimax algorithm can be enhanced by a technique similar to alpha-beta pruning for deterministic game tree search. This involves correctly propagating the $[\alpha, \beta]$ bounds and performing an additional pruning step at each chance node. This pruning step is based on the simple observation that if the minimax value has already been computed for a subset of successors $\tilde{\mathcal{S}} \subset \mathcal{S}$, the minimax value of the state-action pair (s,a) must lie within

$$L_d(s,a) \leq V_d(s,a) \leq U_d(s,a),$$

where

$$L_d(s,a) = \sum_{s' \in \tilde{\mathcal{S}}} \mathcal{P}(s' \mid s,a) V_{d-1}(s') + \sum_{s' \in \mathcal{S} \setminus \tilde{\mathcal{S}}} \mathcal{P}(s' \mid s,a) v_{\min}$$

$$U_d(s,a) = \sum_{s' \in \tilde{\mathcal{S}}} \mathcal{P}(s' \mid s,a) V_{d-1}(s') + \sum_{s' \in \mathcal{S} \setminus \tilde{\mathcal{S}}} \mathcal{P}(s' \mid s,a) v_{\max}.$$

These bounds form the basis of the pruning mechanisms in the *-Minimax [7] family of algorithms. In the Star1 algorithm, each s' from the equations above represents the state reached after a particular outcome is applied at a chance node following (s,a). In practice, Star1 maintains lower and upper bounds on $V_{d-1}(s')$ for each child s' at chance nodes,

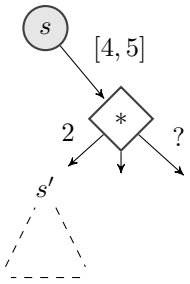


Figure 6.1: An example of the STAR1 algorithm.

using this information to stop the search when it finds a proof that any future search is pointless.

To better understand when cutoffs occur in *-Minimax, we now present an example adapted from Ballard's original paper. Consider Figure 6.1. The algorithm recurses down from state s with a window of $[\alpha, \beta] = [4, 5]$ and encounters a chance node. Without having searched any of the children the bounds for the values returned are $(v_{\min}, v_{\max}) = (-10, +10)$. The subtree of a child, say s', is searched and returns $V_{d-1}(s') = 2$. Since this is now known, the upper and lower bounds for that outcome become 2. The lower bound on the minimax value of the chance node becomes $(2 - 10 - 10)/3$ and the upper bound becomes $(2 + 10 + 10)/3$, assuming a uniform distribution over chance events. If ever the lower bound on the value of the chance node exceeds β, or if the upper bound for the chance node is less than α, the subtree is pruned. In addition, this bound information is used to compute new bounds to send to the other child nodes.

The Star1 algorithm is summarized in Algorithm 6. Recall from Section 2.3.2 that we always treat player 1 as the player maximizing $u_1(s)$ (*Max*), and player 2 as the player minimizing $u_1(s)$ (*Min*). The parameter c is a boolean representing whether or not a chance node is the next node in the tree[1]. The outcome set o is an array of tuples, one per outcome. The ith tuple has three attributes: a lower bound o_{il} initialized to $v_{\min}$, an upper bound o_{iu} initialized to $v_{\max}$, and the outcome's probability o_{ip}. The lowerBound function returns the current lower bound on the chance node $\sum_{i \in \{0,\ldots,N-1\}} o_{ip} o_{il}$. Similarly, upperBound returns the current upper bound on the chance node using o_{iu} in place of o_{il}. Finally, the functions computeChildAlpha and computeChildBeta return the new bounds on the value of the respective child below. Continuing the example above, suppose the algorithm is ready to descend down the middle outcome. The lower bound for the child is derived from the

[1]Note that we assume that chance nodes and decision nodes alternate

Algorithm 6 Star1

```
1: alphabeta1(s, d, α, β)
2:     if d = 0 or s ∈ Z then return (h_1(s), null)
3:     else if P(s) = 1 (Max) then
4:         α' ← α
5:         (v*, a*) ← (−∞, null)
6:         for a ∈ A(s) do
7:             v ← Star1(s, a, d − 1, α', β, true)
8:             if v ≥ v* then (v*, a*, α') ← (v, a, max(v, α'))
9:             if β ≤ α' break
10:        return (v*, a*)
11:    else if P(s) = 2 (Min) then
12:        β' ← β
13:        (v*, a*) ← (+∞, null)
14:        for a ∈ A(s) do
15:            v ← Star1(s, a, d − 1, α, β', true)
16:            if v ≤ v* then (v*, a*, β') ← (v, a, min(v, β'))
17:            if β' ≤ α break
18:        return (v*, a*)
19:
20: Star1(s, a, d, α, β, c)
21:    if d = 0 or s ∈ Z then return (h_1(s), null)
22:    else if ¬c then return alphabeta1(s, d, α, β)
23:    else
24:        o ← genOutcomeSet(s, a, v_min, v_max)
25:        N ← |o|
26:        for i ∈ {0, ..., N − 1}
27:            α' ← computeChildAlpha(o, α, i); β' ← computeChildBeta(o, β, i)
28:            s' ← applyActionAndChanceOutcome(s, a, i)
29:            (v, a') ← Star1(s', null, d − 1, max(v_min, α'), min(v_max, β'), false)
30:            o_il ← v; o_iu ← v
31:            if v ≥ β' then return (lowerBound(o), null)
32:            if v ≤ α' then return (upperBound(o), null)
33:        return (exactValue(o), null)
```

equation $(2+o_{1p}\alpha'+10)/3 = \alpha$. Solving for α' here gives $\alpha' = (3\alpha-12)/o_{1p}$. In general:

$$\alpha' = \frac{\alpha - \text{upperBound}(o) + o_{ip}o_{iu}}{o_{ip}}, \qquad \beta' = \frac{\beta - \text{lowerBound}(o) + o_{ip}o_{il}}{o_{ip}}.$$

The performance of the algorithm can be improved significantly by applying a simple look-ahead heuristic. Suppose the algorithm encounters a chance node. When searching the children of each outcome, one can temporarily restrict the legal actions at a successor (decision) node. If only a single action is searched at the successor, then the value returned will be a bound on $V_{d-1}(s')$. If the successor is a Max node, then the true value can only be larger, and hence the value returned is a lower bound. Similarly, if it was a Min

Algorithm 7 Star2

```
1:  Star2(s, a, d, α, β, c, p)
2:    if d = 0 or s ∈ Z then return (h_1(s), null)
3:    else if ¬c then return alphabeta2(s, d, α, β, p)
4:    else
5:      o ← genOutcomeSet(s, a, v_min, v_max)
6:      N ← |o|
7:      for i ∈ {0, ..., N − 1}
8:        α' ← computeChildAlpha(o, α, i); β' ← computeChildBeta(o, β, i)
9:        s' ← applyActionAndChanceOutcome(s, a, i)
10:       (v, a') ← Star2(s', null, d − 1, max(v_min, α'), min(v_max, β'), false, true)
11:       if ¶(s') = 1 then
12:         o_il ← v
13:         if lowerBound(o) ≥ β then return (lowerBound(o), null)
14:       else if P(s') = 2 then
15:         o_iu ← v
16:         if upperBound(o) ≤ α then return (upperBound(o), null)
17:     for i ∈ {0, ..., N − 1}
18:       α' ← computeChildAlpha(o, α, i); β' ← computeChildBeta(o, β, i)
19:       s' ← applyActionAndChanceOutcome(s, a, i)
20:       (v, a') ← Star2(s', null, d − 1, max(v_min, α'), min(v_max, β'), false, false)
21:       o_il ← v; o_iu ← v
22:       if v ≥ β' then return (lowerBound(o), null)
23:       if v ≤ α' then return (upperBound(o), null)
24:     return (exactValue(o), null)
```

node, the value returned is an upper bound. The Star2 algorithm applies this idea via a preliminary *probing phase* at chance nodes in hopes of pruning without requiring full search of the children. If probing does not lead to a cutoff, then the children are fully searched, but bound information collected in the probing phase can be re-used. When moves are appropriately ordered, the algorithm can often choose the best single move and effectively cause a cut-off with exponentially less search effort. Since this is applied recursively, the benefit compounds as the depth increases. The algorithm is summarized in Algorithm 7. The alphabeta2 procedure is analogous to alphabeta1 except when p is true, a subset (of size one) of the actions are considered at the next decision node. The recursive calls to Star2 within alphabeta2 have have p set to false and a set to the chosen action.

Note that Star1 and Star2 are typically presented using the negamax formulation[2]. In fact, Ballard originally restricted his discussion to regular *-minimax trees, ones that strictly alternate Max, Chance, Min, Chance. We intentionally present the more general $\alpha - \beta$

[2]In the negamax formulation, instead of having two separate cases (one for *Max* and one for *Min*) as seen in in lines 3 to 18 of Algorithm 6, there is only a single case: always maximize. The values are negated at each recursive call so that when *Min* is to act, they are maximizing the negation of the payoff to Max.

formulation here because it handles a specific case, where *Max* and *Min* do not necessarily alternate, encountered in two of our three test domains. For example, in two of our domains we can observe a sequence: Max, Chance, Max, Chance.

In games where the outcome of a chance node determines the next player to play, the cut criteria during the Star2 probing phase depends on the child node. The bound established by the Star2 probing phase will either be a lower bound or an upper bound, depending on the child's type. This distinction is made in lines 11 to 16. Also note: when implementing the algorithm, we have found better performance occurs when incrementally computing the bound information [44].

6.2 Related Work

Before describing our approach, we will give an overview of two related algorithms that have been applied in these settings.

6.2.1 MCTS with Double-Progressive Widening

Recall Monte Carlo Tree Search from Section 2.3.2. In recent years Monte Carlo methods have seen a surge of popularity in tree search methods for games. An improvement of practical importance has been established called Progressive Unpruning / Widening [21; 19]. The main idea here is to purposely restrict the number of actions; this width is gradually increased so that the tree grows deeper at first and then slowly wider over time.

The progressive widening idea is extended to include chance nodes in the double progressive widening algorithm (MCTS+DPW) [20]. When MCTS+DPW encounters a chance or decision node, it computes a maximum number of actions or outcomes to consider $k = \lceil Cv^{\alpha} \rceil$, where C and α are parameter constants and v represents a number of visits to the node. At a decision node, then only the first k actions from the action set are available. At a chance node, a set of outcomes is stored and incrementally grown. An outcome is sampled; if k is larger than the size of the current set of outcomes and the newly sampled outcome is not in the set, it is added to the set. When the branching factor at chance nodes is extremely high, double progressive widening prevents MCTS from degrading into 1-ply rollout planning. We will use MCTS enhanced with double progressive widening as one of the baseline algorithms for our experimental comparison.

6.2.2 Sampling Methods for Markov Decision Processes

Computing optimal policies in large Markov Decision Processes (MDPs) is a significant challenge. Since the size of the state space is often exponential in the number of properties describing each state, much work has focused on finding efficient methods to compute approximately optimal solutions. One way to do this, given only a generative model of the domain, is to employ *sparse sampling* [58]. When faced with a decision to make from a particular state, a local sub-MDP is built using finite horizon look-ahead search. When transitioning to successor states, a fixed number $c \in \mathbb{N}$ of successor states are sampled for each action. Kearns et al. showed that for an appropriate choice of c, this procedure produces value estimates that are accurate (with high probability). Importantly, c was shown to have no dependence on the number of states $|\mathcal{S}|$, effectively breaking the curse of dimensionality.

This method of sparse sampling was improved by using adaptive decision rules based on the multi-armed bandit literature to give the AMS algorithm [16]. Also, the Forward Search Sparse Sampling (FSSS) [115] algorithm was recently introduced, which exploits bound information to add a form of sound pruning to sparse sampling. The pruning mechanism used by FSSS is analogous to what Star1 performs in adversarial domains.

We will now look at adapting these ideas in the game tree search setting.

6.3 Sparse Sampling in Adversarial Games

The performance of classical game tree search also suffers from a dependence on $|\mathcal{S}|$. Like Sparse Sampling for MDPs [58], we remove this dependence using Monte-Carlo sampling.

We define the *estimated finite horizon minimax value* as

$$\hat{V}_d(s) = \begin{cases} \max\limits_{a \in \mathcal{A}} \hat{V}_d(s,a) & \text{if} \quad d > 0, s \notin Z, \text{ and } P(s) = 1 \\ \min\limits_{a \in \mathcal{A}} \hat{V}_d(s,a) & \text{if} \quad d > 0, s \notin Z, \text{ and } P(s) = 2 \\ h(s) & \text{otherwise.} \end{cases} \tag{6.1}$$

where

$$\hat{V}_d(s,a) = \tfrac{1}{c} \sum_{i=1}^{c} \hat{V}_{d-1}(s_i),$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, with each successor state $s_i \sim \mathcal{P}(\cdot \mid s, a)$ for $1 \leq i \leq c$. Our main objective is to prove that for a sufficiently large value of c, these estimates are accurate. Before doing so, we present a few building blocks. The non-trivial proofs of these supporting lemmas and propositions are included in Appendix A.3.

Lemma 7. *For all states $s \in \mathcal{S}$, for all actions $a \in \mathcal{A}$, for all $\lambda \in (0, 2v_{\max}] \subset \mathbb{R}$, for all $c \in \mathbb{N}$, given a set $\mathcal{C}(s)$ of $c \in \mathbb{N}$ states generated according to $\mathcal{P}(\cdot \mid s, a)$, we have*

$$\mathbb{P}\left(\left|\left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i)\right] - V_d(s,a)\right| \geq \lambda\right) \leq 2\exp\left\{-\lambda^2 c \;/\; 2v_{\max}^2\right\}. \tag{6.2}$$

Proposition 1. *For all $d \in \mathbb{N}$, for a state $s \in \mathcal{S}$, if $\left|\hat{V}_d(s,a) - V_d(s,a)\right| < \lambda$ holds for all $a \in \mathcal{A}$, then $\left|\hat{V}_d(s) - V_d(s)\right| < \lambda$.*

We now state our main result.

Theorem 8. *Given $c \in \mathbb{N}$, for any state $s \in \mathcal{S}$, for all $\lambda \in (0, 2v_{\max}] \subset \mathbb{R}$, for any depth $d \in \mathbb{Z}_+$,*

$$\mathbb{P}\left(\left|\hat{V}_d(s) - V_d(s)\right| \leq \lambda d\right) \geq 1 - (2c|\mathcal{A}|)^d \exp\left\{-\lambda^2 c \;/\; 2v_{\max}^2\right\}.$$

Proof. We will use an inductive argument. First note that the base case is trivially satisfied for $d = 0$, since $\hat{V}_0(s) = V_0(s)$ for all $s \in \mathcal{S}$ by definition. Now, assume that the statement is true for some $d - 1 \in \mathbb{Z}_+$ i.e.

$$\mathbb{P}\left(\left|\hat{V}_{d-1}(s) - V_{d-1}(s)\right| \leq \lambda(d-1)\right) \geq 1 - (2c|\mathcal{A}|)^{d-1} \exp\left\{-\lambda^2 c \;/\; 2v_{\max}^2\right\}. \tag{6.3}$$

Next we bound the error for each state-action estimate $\hat{V}_d(s,a)$. We denote by $\mathcal{C}(s) \subseteq \mathcal{S}$ the set of $c \in \mathbb{N}$ successor states drawn from $\mathcal{P}(\cdot \mid s, a)$. So, $\left|\hat{V}_d(s,a) - V_d(s,a)\right|$

$$\begin{aligned}
&= \left|\left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} \hat{V}_{d-1}(s_i)\right] - V_d(s,a)\right| \\
&= \left|\left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} \hat{V}_{d-1}(s_i)\right] - \left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i)\right] + \left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i)\right] - V_d(s,a)\right| \\
&\leq \frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} \left|\hat{V}_{d-1}(s_i) - V_{d-1}(s_i)\right| + \left|\left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i)\right] - V_d(s,a)\right| \qquad (6.4)
\end{aligned}$$

The first step follows from the definition of $\hat{V}_d(s,a)$. The final step follows by the fact that $|a-b| \leq |a-c| + |c-b|$ and simplifying. The RHS of Equation (6.4) consists of a sum of two terms, which we analyze in turn. The first term

$$\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} \left|\hat{V}_{d-1}(s_i) - V_{d-1}(s_i)\right|$$

is the average of the error in c state value estimates at level $d-1$. Now, the event that

$$\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} \left| \hat{V}_{d-1}(s_i) - V_{d-1}(s_i) \right| > \lambda(d-1)$$

is a subset of the event that a single estimate is off by more than $\lambda(d-1)$. Therefore, we have

$$\begin{aligned}
&\mathbb{P}\left(\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} \left| \hat{V}_{d-1}(s_i) - V_{d-1}(s_i) \right| > \lambda(d-1) \right) \\
&\leq \mathbb{P}\left(\bigcup_{s_i \in \mathcal{C}} \left| \hat{V}_{d-1}(s_i) - V_{d-1}(s_i) \right| > \lambda(d-1) \right) \\
&\leq \sum_{s_i \in \mathcal{C}} \mathbb{P}\left(\left| \hat{V}_{d-1}(s_i) - V_{d-1}(s_i) \right| > \lambda(d-1) \right) \\
&\leq c\left(2c|\mathcal{A}|\right)^{d-1} \exp\left\{ -\lambda^2 c \;/\; 2v_{\max}^2 \right\}. \qquad (6.5)
\end{aligned}$$

The penultimate line follows from the union bound. The final line applies the inductive hypothesis.

We now consider the second term

$$\left| \left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i) \right] - V_d(s,a) \right|$$

of the RHS of Equation (6.4). By Lemma 7 we have

$$\mathbb{P}\left(\left| \left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s) \right] - V_d(s,a) \right| > \lambda \right) \leq 2\exp\left\{ -\lambda^2 c \;/\; 2v_{\max}^2 \right\}. \qquad (6.6)$$

We now have a bound for each of the two terms in the RHS of Equation (6.4), as well as the probability with which that bound is exceeded. Notice that the value of the RHS can exceed the sum of the two terms' bounds if either term exceeds its respective bound. Using this, we get

$$\mathbb{P}\left(\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} \left| \hat{V}_{d-1}(s_i) - V_{d-1}(s_i) \right| + \left| \left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i) \right] - V_d(s,a) \right| > \lambda d \right) \leq$$

$$\mathbb{P}\left(\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} \left| \hat{V}_{d-1}(s_i) - V_{d-1}(s_i) \right| > \lambda(d-1) \right) + \mathbb{P}\left(\left| \left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i) \right] - V_d(s,a) \right| > \lambda \right)$$

by the union bound and the fact that $x + y \geq K \Rightarrow x \geq k_1$ or $y \geq k_2$, where $K = k_1 + k_2$; specifically, the event on the left-hand side is a subset of the union of the two events on the right-hand side.

Continuing, we can apply Equations 6.5 and 6.6 to the above to get an upper bound of

$$\begin{aligned} & c\left(2c|\mathcal{A}|\right)^{d-1}\exp\left\{-\lambda^2 c \,/\, 2v_{\max}^2\right\} + 2\exp\left\{-\lambda^2 c \,/\, 2v_{\max}^2\right\} \\ = \;& \left(2 + c\left(2c|\mathcal{A}|\right)^{d-1}\right)\exp\left\{-\lambda^2 c \,/\, 2v_{\max}^2\right\} \\ \leq \;& (2c)^d\left(|\mathcal{A}|\right)^{d-1}\exp\left\{-\lambda^2 c \,/\, 2v_{\max}^2\right\}, \end{aligned} \tag{6.7}$$

where the final two lines follow from standard calculations and the fact that $c > 1$. Recall from Equation 6.4 that

$$\left|\hat{V}_d(s,a) - V_d(s,a)\right| \leq \frac{1}{c}\sum_{s_i \in \mathcal{C}(s)}\left|\hat{V}_{d-1}(s_i) - V_{d-1}(s_i)\right| + \left|\left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i)\right] - V_d(s,a)\right|$$

This allows us to bound the probability that $\hat{V}_d(s,a)$ differs from $V_d(s,a)$ by more than λd, by using Equation 6.7 as follows

$$\begin{aligned} & \mathbb{P}\left(\left|\hat{V}_d(s,a) - V_d(s,a)\right| > \lambda d\right) \\ \leq \;& \mathbb{P}\left(\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)}\left|\hat{V}_{d-1}(s_i) - V_{d-1}(s_i)\right| + \left|\left[\frac{1}{c}\sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i)\right] - V_d(s,a)\right| > \lambda d\right) \\ \leq \;& (2c)^d\left(|\mathcal{A}|\right)^{d-1}\exp\left\{-\lambda^2 c \,/\, 2v_{\max}^2\right\} \end{aligned} \tag{6.8}$$

We know from Proposition 1 that if all of the chance node value estimates are accurate then the decision node estimate must also be accurate. This allows us to consider the probability of the event that at least one of the chance node value estimates $\hat{V}_1(s,a)$ deviates by more than λd, that is,

$$\mathbb{P}\left(\bigcup_{a \in \mathcal{A}}\left|\hat{V}_d(s,a) - V_d(s,a)\right| > \lambda d\right) \leq \sum_{a \in \mathcal{A}}\mathbb{P}\left(\left|\hat{V}_d(s,a) - V_d(s,a)\right| > \lambda d\right)$$

by the union bound. Applying Equation 6.8, we get

$$\mathbb{P}\left(\bigcup_{a \in \mathcal{A}}\left|\hat{V}_d(s,a) - V_d(s,a)\right| > \lambda d\right) \leq \left(2c|\mathcal{A}|\right)^d\exp\left\{-\lambda^2 c \,/\, 2v_{\max}^2\right\},$$

hence

$$1 - \mathbb{P}\left(\bigcup_{a \in \mathcal{A}}\left|\hat{V}_d(s,a) - V_d(s,a)\right| > \lambda d\right) \geq 1 - \left(2c|\mathcal{A}|\right)^d\exp\left\{-\lambda^2 c \,/\, 2v_{\max}^2\right\}.$$

Then by De Morgan's law, we have

$$\mathbb{P}\left(\bigcap_{a \in \mathcal{A}}\left|\hat{V}_d(s,a) - V_d(s,a)\right| \leq \lambda d\right) \geq 1 - \left(2c|\mathcal{A}|\right)^d\exp\left\{-\lambda^2 c \,/\, 2v_{\max}^2\right\},$$

which combined with Proposition 1 implies that

$$\mathbb{P}\left(\left|\hat{V}_d(s) - V_d(s)\right| \leq \lambda d\right) \geq 1 - (2c|\mathcal{A}|)^d \exp\left\{-\lambda^2 c \;/\; 2v_{\max}^2\right\},$$

which proves the inductive step. □

The proof is a straightforward generalization of the result of [58] for finite horizon, adversarial games. The theorem shows that the higher the desired accuracy of the estimate, the lower one must choose λ, in turn increasing the value that must be chosen for the sample width c. For a particular tolerance parameter λ and depth d: as c grows, the probability of $\hat{V}_d(s)$ being an accurate estimate of the true value $V_d(s)$ approaches 1.

The MCMS variants can be easily described in terms of the descriptions of Star1 and Star2. To enable sampling, one need only change the implementation of getOutcomeSet on line 24 of Algorithm 6 and line 5 of Algorithm 7. Instead of generating the full list of moves, the new function samples c outcomes *with replacement* and assigns a uniform distribution over the new outcome set of size c. We call these new variants Star1SS and Star2SS. If all pruning is disabled, we obtain Expectimax with sparse sampling (ExpSS), which computes $\hat{V}_d(s)$ directly from the definition in Equation 6.1. The Star1SS method computes exactly the same value as ExpSS, but can avoid useless work. The same can be said for Star2SS, provided exactly the same set of chance events is used whenever a state-action pair is visited; this additional restriction is needed due to the extra probing phase in Star2. Note that while sampling without replacement may work better in practice, the proof of Theorem 8 relies on the Hoeffding bound, which requires sampling with replacement.

6.4 Experiments

In this section, we present an empirical evaluation of MCMS.

Recall the descriptions of Pig, EinStein Würfelt Nicht! (EWN), and Can't Stop from Section 3.2. At least one MCTS player has been developed to play EWN [74].

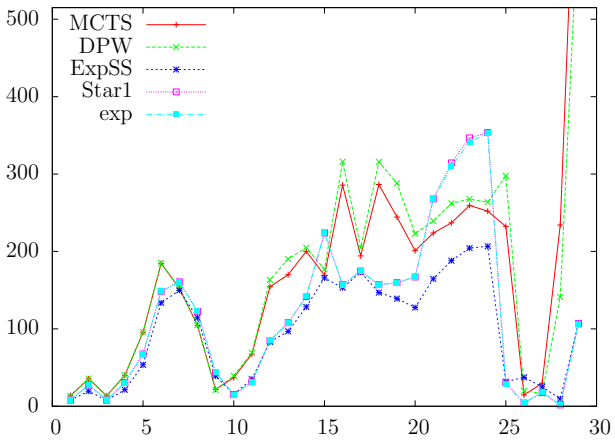
To evaluate our algorithm, we performed two separate experiments. In all of our experiments, a time limit of 0.1 seconds of search is used. MCTS uses utilities in $[-100, 100]$ and a tuned exploration constant value of 50. To make a more direct comparison to MCMS, MCTS returns the value of the heuristic evaluation function at the leaves rather than using a rollout policy. MCTS with double-progressive widening (DPW) uses parameters C and α described in Section 2.3.2. All experiments were single-threaded and run on the same hardware (equipped with Intel Core i7 3.4Ghz processors). The best sample widths for ExpSS,

Star1SS, Star2SS, and (C, α) for DPW for Pig were $(20, 2, 8, (5, 0.4))$. For EWN and Can't Stop these parameters were set to $(5, 2, 4, (3, 0.05))$ and $(5, 50, 18, (10, 0.2))$ respectively. These values were determined by running a number of round-robin tournaments between players of the same type. The precise effect of the value of the sample width parameter on the quality of the MCMS algorithm is not included here. We also suspect that a dynamically-determined width (dependent on the current depth) may help. We leave both of these items as potential future work.

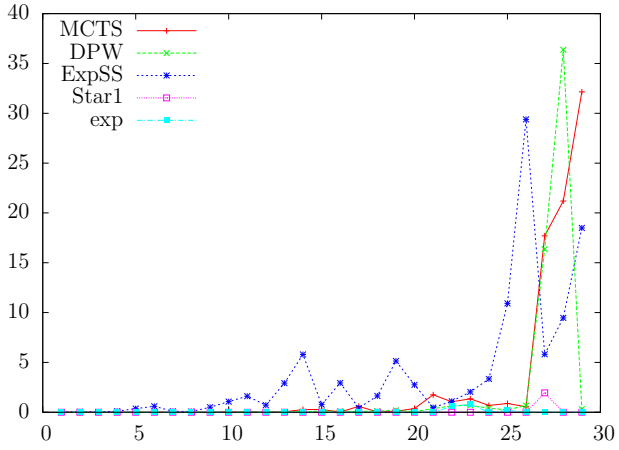
Our first experiment compares statistical properties of the estimates returned and actions recommended by MCMS and MCTS. At a decision point, each algorithm returns a recommended move $a \in \mathcal{A}$ and acts as an estimator of its minimax value $\hat{V}(s)$. Since Pig has fewer than one million states, we solve it using the technique of value iteration which has been applied to previous smaller games of Pig [78], obtaining the true value of each state $V(s)$. From this, we estimate the *mean squared error*, *variance*, and *bias* of each algorithm: $\text{MSE}[\hat{V}(s)] = \mathbb{E}[(\hat{V}(s) - V(s))^2] = \mathbb{V}\text{ar}[\hat{V}(s)] + \text{Bias}(V(s), \hat{V}(s))^2$ by taking 30 samples of each algorithm at each decision point. We define the regret of taking action a at state s to be $\text{Regret}(s, a) = V(s) - V(s, a)$, where a is the action chosen by the algorithm from state s. We measure the average value of $\text{Regret}(s, a)$ over the 30 samples at each decision point for each algorithm. This experiment was performed on several games of Pig; the results are shown in Figures 6.2, 6.3, and 6.4. We intentionally exclude a single plot representing averages over many games because the locations of the peaks are different in each game, making the plot difficult to read.

From the results of this first experiment, we see that the estimated bias of the values returned by ExpSS is generally lower than both MCTS and the non-sampling algorithms. The sample based estimates are particularly more accurate from turns 18 to 25, possibly due to the fact that the ExpSS is reaching the leaves and returning a true value while the others are not. The reduction in bias comes at cost of higher variance value estimates, especially toward the end of the game. But, when combined, ExpSS shows lower mean squared error; since the MSE grows quadratically in the bias (as opposed to linearly in the variance), reducing the bias by just a smaller amount than the increase in variance can still result in a more accurate estimator. Due to this increased variance, one potential future investigation could be to apply one or more variance reduction techniques described in Chapter 7. ExpSS also often exhibits lower regret than expectimax and Star1 but not always lower than MCTS.

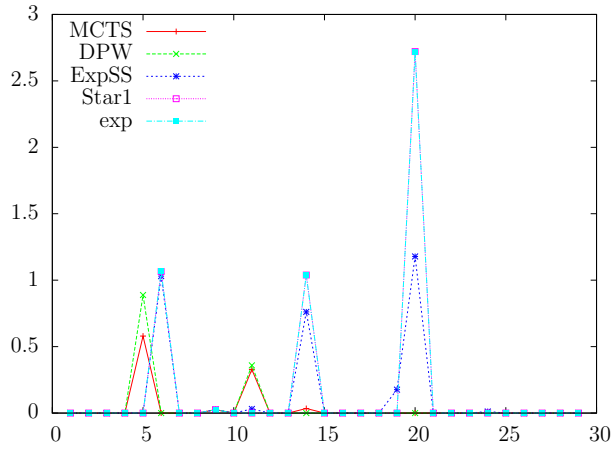
In our second experiment, we computed the performance of each algorithm by playing 500 test matches for each paired set of players. Each match consists of two games where



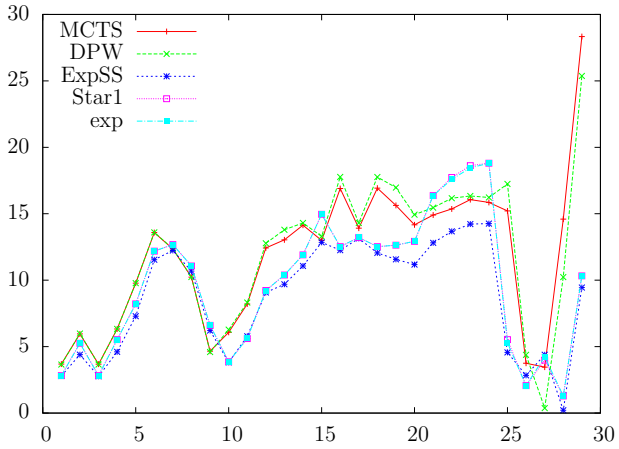
(a) MSE of returned value in Game #1



(b) Variance of returned value in Game #1

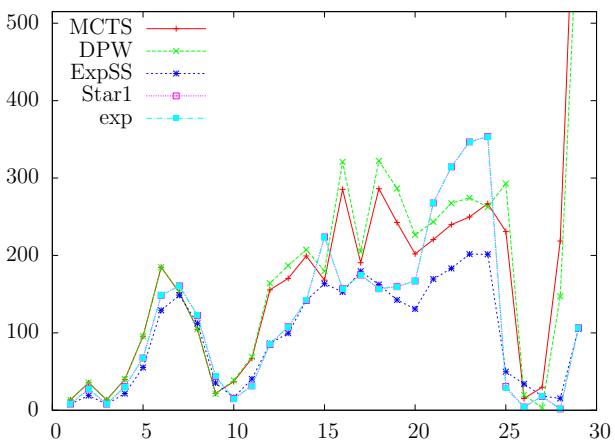


(c) Regret of returned move in one Game #1

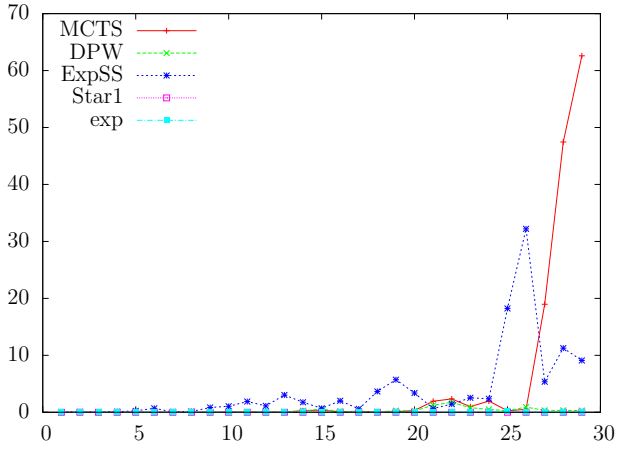


(d) Bias of returned value in one Game #1

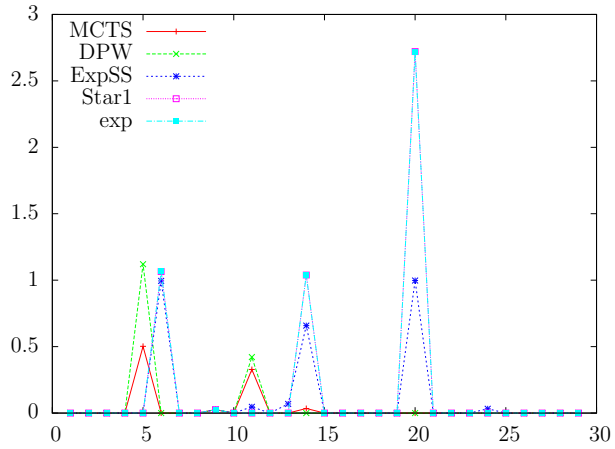
Figure 6.2: Properties of MCMS on Pig (Game #1). Exp and ExpSS represent EXPECTIMAX without and with sparse sampling, respectively. DPW represents MCTS with double-progressive widening.



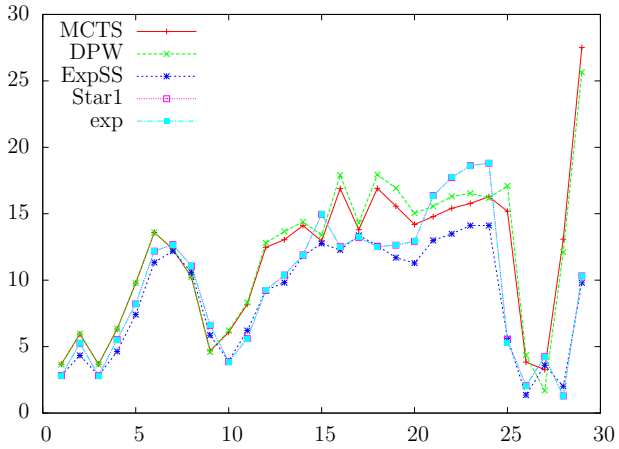
(a) MSE of returned value in Game #2



(b) Variance of returned value in Game #2

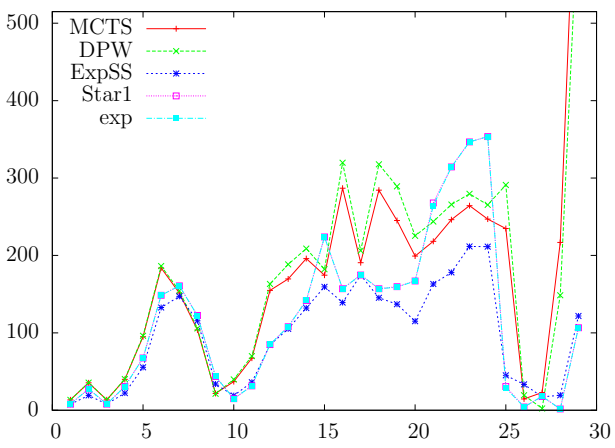


(c) Regret of returned move in one Game #2

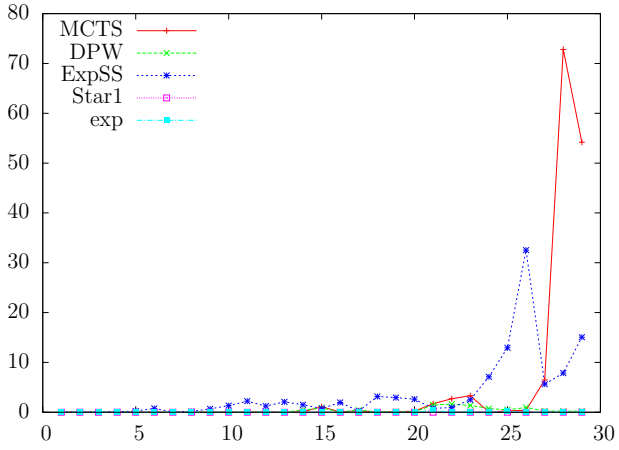


(d) Bias of returned value in one Game #2

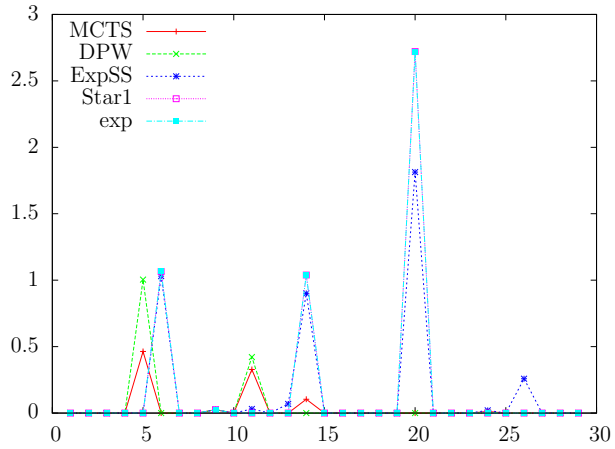
Figure 6.3: Properties of MCMS on Pig (Game #2). Exp and ExpSS represent EXPECTIMAX without and with sparse sampling, respectively. DPW represents MCTS with double-progressive widening.



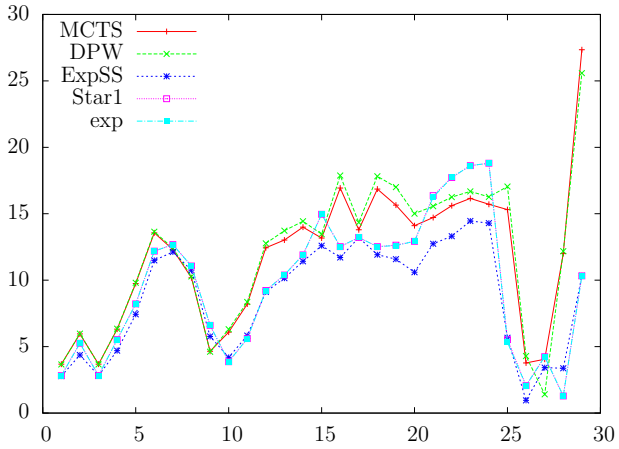
(a) MSE of returned value in Game #3



(b) Variance of returned value in Game #3



(c) Regret of returned move in one Game #3



(d) Bias of returned value in one Game #3

Figure 6.4: Properties of MCMS on Pig (Game #3). Exp and ExpSS represent EXPECTIMAX without and with sparse sampling, respectively. DPW represents MCTS with double-progressive widening.

	ExpSS-Exp	Star1SS-Star1	Star2SS-Star2	
Pig	55.5	51.4	48.5	
EWN	51.9	50.7	50.1	
Can't Stop	81.1	84.4	84.6	
	ExpSS-DPW	Star1SS-DPW	Star2SS-DPW	MCTS-DPW
Pig	49.9	40.5	46.1	48.3
EWN	49.7	45.2	50.1	51.3
Can't Stop	80.4	80.1	77.9	18.9

Table 6.1: Win percentage for p_1 in a p_1-p_2 match of 1000 games in Pig (Pig Out), EWN, and Can't Stop.

players swap seats and a single random seed is generated and used for both games in the match. The performance of each pairing of players is shown in Table 6.1.

The results from Table 6.1 show that the MCMS variants outperform their equivalent non-sampling counterparts in all but one instance (Star2SS vs. Star2 in Pig); this might be explained by the fact that since there are only 2 actions to choose from in Pig, it is easy to determine a move ordering so that the Star2 probing phase works well enough without the need for sampling. DPW also outperformed MCMS on Pig. This was somewhat expected since it exhibited lower regret from the first experiment. In EWN, MCMS performs evenly with DPW. Of the MCMS algorithms, Star2SS does best in EWN, likely due to the strictly alternating roles which give higher chances for cutoffs to occur during the Star2 probing phase. Finally, we see that MCMS wins by large margins in Can't Stop, the domain with the largest branching factor at chance nodes. This suggests that MCMS is well suited for densely stochastic games.

6.5 Chapter Summary and Conclusion

In this chapter we present MCMS, a Monte Carlo sampling algorithm based on expectimax and Ballard's *-Minimax. We have shown that, in theory, the estimates computed by MCMS converge to the true Minimax values as the sample width c tends to infinity. In Pig, we observe that this can increase variance toward the end of the game, but decrease the bias and mean squared error in critical parts of the game. In medium-sized games, MCMS is shown to compete with the state of the art, performing particularly well in Can't Stop, the largest of the three games.

Chapter 7

Variance Reduction Techniques

While sampling techniques aim to reduce computation time by estimating quantities of interest, the success of the approach depends critically on the variance of the estimators. Suppose an algorithm needs to compute a quantity v, such as a counterfactual value in CFR. Computing this value may take a long time, as in Vanilla CFR. Now suppose that we now define two unbiased estimators for this quantity $\hat{v}_1$ and $\hat{v}_2$. Clearly, we want the most precise value of v possible; if both estimators save the same amount of computation time when estimating v then the more precise estimator (one with lower variance) is preferred. Under ideal conditions the estimator would have zero variance which would lead to a direct increase in performance. As this is rare in practice, we can ask ourselves how variance can be reduced and how beneficial it is to reduce this variance.

Often there are ways to reduce the variance of an estimator by injecting some domain knowledge. In this chapter, we survey some common methods used in the Monte Carlo sampling literature. In particular, we describe the application of these *variance reduction techniques* to Monte Carlo sampling. We will focus our application of each variance reduction technique to the single-player Monte Carlo Tree Search (MCTS) setting, as described in Section 2.3.2. We will also show two straight-forward applications of these techniques to MCCFR in Section 7.5.

We examine three variance reduction techniques: control variates, common random numbers and antithetic variates. Each subsection begins with a short overview of each variance reduction technique, followed by a description of how MCTS can be modified to efficiently incorporate it. Each technique is evaluated in practice on three solitaire variants of Pig, Can't Stop, and Dominion. In all cases, solitaire variants of the games are used where the aim is to maximize the number of points given a fixed number of turns. The work presented in this chapter is an extended version of the work presented at the Twenty-Fifth

Annual Conference on Neural Information Processing Systems (NIPS 2011) [110], done in collaboration with Joel Veness and Michael Bowling.

In this chapter, given an independent and identically distributed sample $(X_1, X_2, \dots X_n)$, the sample mean is denoted by $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$. Provided $\mathbb{E}[X]$ exists, $\bar{X}$ is an unbiased estimator of $\mathbb{E}[X]$ with variance $\mathbb{V}\text{ar}[X]/n$.

7.1 Control Variates

An improved estimate of $\mathbb{E}[X]$ can be constructed if we have access to an additional statistic Y that is correlated with X, provided that $\mu_Y = \mathbb{E}[Y]$ exists and is known. To see this, note that if $Z = X + c(Y - \mathbb{E}[Y])$, then $\bar{Z}$ is an unbiased estimator of $\mathbb{E}[X]$, for any $c \in \mathbb{R}$. Y is called the *control variate*. One can show that $\mathbb{V}\text{ar}[Z]$ is minimized when $c = c^*$, where $c^* = -\mathbb{C}\text{ov}[X, Y]/\mathbb{V}\text{ar}[Y]$. Given a sample $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ and setting $c = c^*$, the control variate enhanced estimator

$$\bar{X}_{cv} = \frac{1}{n}\sum_{i=1}^{n} [X_i + c^*(Y_i - \mu_Y)] \tag{7.1}$$

is obtained, with variance

$$\mathbb{V}\text{ar}[\bar{X}_{cv}] = \frac{1}{n}\left(\mathbb{V}\text{ar}[X] - \frac{\mathbb{C}\text{ov}[X, Y]^2}{\mathbb{V}\text{ar}[Y]}\right).$$

Thus the total variance reduction is dependent on the strength of correlation between X and Y. For the optimal value of c, the variance reduction obtained by using Z in place of X is $100 \times \mathbb{C}\text{orr}[X, Y]^2$ percent. In practice, both $\mathbb{V}\text{ar}[Y]$ and $\mathbb{C}\text{ov}[X, Y]$ are unknown and need to be estimated from data. One solution is to use the plug-in estimator $C_n = -\widehat{\mathbb{C}\text{ov}}[X, Y]/\widehat{\mathbb{V}\text{ar}}(Y)$, where $\widehat{\mathbb{C}\text{ov}}[\cdot, \cdot]$ and $\widehat{\mathbb{V}\text{ar}}(\cdot)$ denote the sample covariance and sample variance respectively. This estimate can be constructed offline using an independent sample or be estimated online. Although replacing c^* with an online estimate of C_n in Equation 7.1 introduces bias, this modified estimator is still consistent [79]. Note that $\bar{X}_{cv}$ can be efficiently computed with respect to C_n by maintaining $\bar{X}$ and $\bar{Y}$ online, since $\bar{X}_{cv} = \bar{X} + C_n(\bar{Y} - \mu_Y)$.

Application to MCTS. Control variates can be applied recursively, by redefining the return $X_{s,a}$ for every state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ to

$$Z_{s,a} = X_{s,a} + c_{s,a}\left(Y_{s,a} - \mathbb{E}[Y_{s,a}]\right), \tag{7.2}$$

provided $\mathbb{E}[Y_{s,a}]$ exists and is known for all $(s,a) \in \mathcal{S} \times \mathcal{A}$, and $Y_{s,a}$ is a function of the random variables $A_t, S_{t+1}, R_{t+1}, \ldots, A_{n-1}, S_n, R_n$ that describe the complete execution of the system after action a is performed in state s. Notice that a separate control variate will be introduced for each state-action pair. Furthermore, as $\mathbb{E}[Z_{s_t,a_t} \mid A_i \sim \pi(\cdot \mid S_i)] = \mathbb{E}[X_{s_t,a_t} \mid A_i \sim \pi(\cdot \mid S_i)]$, for all policies π, for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$ and for all $t < i < n$, the inductive argument used to establish the asymptotic consistency of UCT still applies when control variates are introduced in this fashion [61].

Finding appropriate control variates whose expectations are known in advance can prove difficult. This situation is further complicated in UCT where we seek a set of control variates $\{Y_{s,a}\}$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. Drawing inspiration from advantage sum estimators [119], we now describe a general class of control variates designed for application in UCT. Given a realization of a random simulation trajectory $S_t = s_t$, $A_t = a_t$, $S_{t+1} = s_{t+1}$, $A_{t+1} = a_{t+1}, \ldots, S_n = s_n$, consider control variates of the form

$$Y_{s_t,a_t} = \textstyle\sum_{i=t}^{n-1} \mathbb{I}[b(S_{i+1})] - \mathbb{P}[b(S_{i+1}) \mid S_i{=}s_i, A_i{=}a_i], \tag{7.3}$$

where $b : \mathcal{S} \to \{\text{true}, \text{false}\}$ denotes a boolean function of state and $\mathbb{I}$ denotes the binary indicator function. We choose this specific definition (rather than the single-step $Y_{s_t,a_t} = \mathbb{I}(S_{i+1})$) since in MCTS the value $X_{s,a}$ is approximated using the results of simulated trajectories from root to leaf passing through$s$, and in the game setting rewards are only given as payoffs at the end of the game. However, the single-step control variate may be a better way to apply control variates in general MDPs. In our case, the expectation

$$\mathbb{E}[Y_{s_t,a_t}] = \textstyle\sum_{i=t}^{n-1} \Big(\mathbb{E}\left[\mathbb{I}\left[b(S_{i+1})\right] \mid S_i{=}s_i, A_i{=}a_i\right] - \mathbb{P}\left[b(S_{i+1}) \mid S_i{=}s_i, A_i{=}a_i\right] \Big) = 0,$$

for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$. Thus, using control variates of this form simplifies the task to specifying a state property that is strongly correlated with the return, such that $\mathbb{P}[b(S_{i+1}) \mid S_i{=}s_i, A_i{=}a_i]$ is known for all $(s_i, a_i) \in (\mathcal{S}, \mathcal{A})$, for all $t \leq i < n$. This considerably reduces the effort required to find an appropriate set of control variates for MCTS, and can naturally be applied recursively in the tree.

When designing a control variates with this form, there are several things to consider. First: the time required by the computation of $b(S_{i+1})$ and $\mathbb{P}[b(S_{i+1}) \mid S_i{=}s_i, A_i{=}a_i]$ should be low. If the computational load added is high, then compute the values offline for each state (or common states), is advisable. Secondly, the correlation of the control variate with the reward should be high as the amount of variance reduced depends on this correlation, as shown in Equation 7.1. Also, when choosing between control variates with

the similar correlation, the ones with lower variance will offer more reduction in variance. Finally, the rollout and selection policies may also have an effect on the quality of the control variate.

Application to Test Domains. Our control variates for our domains have the form specified by Equation 7.3. In Pig, we use a boolean function that returns true if we have just performed the roll action and obtain at least one ⚀. This control variate has an intuitive interpretation, since we would expect the return from a single trajectory to be an underestimate if it contained more rolls with a ⚀ than expected, and an overestimate if it contained less rolls with a ⚀ than expected. In Can't Stop, we used similarly inspired boolean function that returned true if we could not make a legal pairing from our most recent roll of the 4 dice. In Dominion, we used a boolean function that returned whether we had just played an action that let us randomly draw a hand with 8 or more money to spend. This is a significant occurrence, as 8 money is needed to buy a Province, the highest and most efficient scoring card in the game. Strong play invariably requires purchasing as many Provinces as possible.

7.2 Common Random Numbers

Consider comparing the expectation of $\mathbb{E}[Y]$ to $\mathbb{E}[Z]$, where both $Y = g(X)$ and $Z = h(X)$ are functions of a common random variable X. This can be framed as estimating the value of $\delta_{Y,Z} = \mathbb{E}[g(X)] - \mathbb{E}[h(X)]$. If the expectations $\mathbb{E}[g(X)]$ and $\mathbb{E}[h(X)]$ were estimated from two independent samples $\mathbf{X}_1$ and $\mathbf{X}_2$, the estimator $\hat{g}(\mathbf{X}_1) - \hat{h}(\mathbf{X}_2)$ would be obtained, with variance $\mathbb{V}\text{ar}[\hat{g}(\mathbf{X}_1) - \hat{h}(\mathbf{X}_2)] = \mathbb{V}\text{ar}[\hat{g}(\mathbf{X}_1)] + \mathbb{V}\text{ar}[\hat{h}(\mathbf{X}_2)]$. Note that no covariance term appears since $\mathbf{X}_1$ and $\mathbf{X}_2$ are independent samples.

Using *common random numbers* suggests setting $\mathbf{X}_1 = \mathbf{X}_2$ if $\mathbb{C}\text{ov}[\hat{g}(\mathbf{X}_1), \hat{h}(\mathbf{X}_2)]$ is positive. This gives the estimator $\hat{\delta}_{Y,Z}(\mathbf{X}_1) = \hat{g}(\mathbf{X}_1) - \hat{h}(\mathbf{X}_1)$, with variance $\mathbb{V}\text{ar}[\hat{g}(\mathbf{X}_1)] + \mathbb{V}\text{ar}[\hat{h}(\mathbf{X}_1)] - 2\mathbb{C}\text{ov}[\hat{g}(\mathbf{X}_1), \hat{h}(\mathbf{X}_1)]$, which is an improvement whenever $\mathbb{C}\text{ov}[\hat{g}(\mathbf{X}_1), \hat{h}(\mathbf{X}_1)]$ is positive. This technique cannot be applied indiscriminately however, since a variance increase will result if the estimates are negatively correlated.

Application to MCTS. Rather than directly reducing the variance of the individual return estimates, common random numbers can instead be applied to reduce the variance of the estimated differences in return $\bar{X}^m_{s,a} - \bar{X}^m_{s,a'}$, for each pair of distinct actions $a, a' \in \mathcal{A}$ in a state s. This has the benefit of reducing the effect of variance in both determining the action $a_t = \arg\max_{a \in \mathcal{A}} \bar{X}^m_{s,a}$ selected by UCT in state s_t and the actions $\arg\max_{a \in \mathcal{A}} \bar{X}^m_{s,a} +$

$c\sqrt{\log(T_s^m)/T_{s,a}^m}$ selected by UCB as the search tree is constructed.

As each estimate $\bar{X}_{s,a}^m$ is a function of realized simulation trajectories originating from state-action pair (s, a), a carefully chosen subset of the stochastic events determining the realized state transitions now needs to be shared across future trajectories originating from s so that $\mathbb{C}\text{ov}[\bar{X}_{s,a}^m, \bar{X}_{s,a'}^m]$ is positive for all $m \in \mathbb{N}$ and for all distinct pairs of actions $a, a' \in \mathcal{A}$. Our approach is to use the same chance outcomes to determine the trajectories originating from state-action pairs (s, a) and (s, a') if $T_{s,a}^i = T_{s,a'}^j$, for any $a, a' \in \mathcal{A}$ and $i, j \in \mathbb{N}$. This can be implemented by using $T_{s,a}^m$ to index into a list of stored stochastic outcomes E_s defined for each state s. By only adding a new outcome to E_s when $T_{s,a}$ exceeds the number of elements in E_s, the list of common chance outcomes can be efficiently generated online.

Application to Test Domains. To apply common random numbers, we need to specify the future chance events to be shared across all of the trajectories originating from each state. Since a player's final score in Pig is strongly dependent on their dice rolls, it is natural to consider sharing one or more future dice roll outcomes. By exploiting the property in Pig that each roll event is independent of the current state, our implementation shares a batch of roll outcomes large enough to drive a complete simulation trajectory. So that these chance events do not conflict, we limited the sharing of roll events to just the root node. A similar technique is used in Can't Stop. We found this scheme to be superior to sharing a single outcome at each state and applying the ideas above recursively. In Dominion, stochasticity is caused by drawing cards from the top of a deck that is periodically shuffled. Here common random numbers are implemented by recursively sharing pre-shuffled deck configurations across the actions at each state. The motivation for this kind of sharing is that it should reduce the chance of one action appearing better than another simply because of "luckier" shuffles.

7.3 Antithetic Variates

Consider estimating the expected value of a function of a random variable $\mathbb{E}[h(X)]$ from two identically distributed samples $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ and $\mathbf{X}' = (X_1', X_2', \ldots, X_n')$. To do this, one can compute $\hat{h}(\mathbf{X}, \mathbf{X}') = \frac{1}{2}[\hat{h}_1(\mathbf{X}) + \hat{h}_2(\mathbf{X}')]$, the average of two unbiased estimates $\hat{h}_1(\mathbf{X})$ and $\hat{h}_2(\mathbf{X}')$. The variance of $\hat{h}(\mathbf{X}, \mathbf{X}')$ is

$$\tfrac{1}{4}(\mathbb{V}\text{ar}[\hat{h}_1(\mathbf{X})] + \mathbb{V}\text{ar}[\hat{h}_2(\mathbf{X}')]) + \tfrac{1}{2}\mathbb{C}\text{ov}[\hat{h}_1(\mathbf{X}), \hat{h}_2(\mathbf{X}')]. \tag{7.4}$$

The method of *antithetic variates* exploits this identity, by deliberately introducing a negative correlation between $\hat{h}_1(\mathbf{X})$ and $\hat{h}_2(\mathbf{X}')$. The usual way to do this is to construct $\mathbf{X}$ and $\mathbf{X}'$ from pairs of sample points (X_i, X_i') such that $\mathbb{C}\text{ov}[h_1(X_i), h_2(X_i')] < 0$ for all $i \leq n$.

For example, consider the following solitaire game. One player has five turns. Each turn, they only have a single action: roll a six-sided die. At the end of a game, the player is awarded a sum a money equal to the sum of the dice rolls. How much money should someone pay to play this game? The mean payoff is useful in answering this question. One can estimated it by taking two samples, *e.g.,* ⚂⚄⚃⚄⚅ and ⚄⚄⚃⚀⚅, and computing the estimated mean payoff to be $\frac{22+18}{2} = 20$. We can augment these two samples with an additional two antithetic samples, *e.g.,* ⚃⚅⚀⚅⚃ and ⚅⚃⚅⚃⚃ giving $\frac{22+18+13+17}{4} = 17.5$, which is a better estimate (in this case, perfect).

In other words, from $\mathbf{X}$ we construct an $\hat{h}_2(\mathbf{X}')$, such that it remains an unbiased estimate of $\mathbb{E}[h(X)]$, and combined with $\hat{h}_1(\mathbf{X})$ gives lower variance estimate. In the example above, this is done by choosing $h_2(X) = 7 - h_1(X)$. Care needs to be taken when constructing such relationships; one must ensure that $\hat{h}_2(\mathbf{X}')$ remains an unbiased estimate of $\mathbb{E}[h(X)]$. In the example above, $\hat{h}_1$ and $\hat{h}_2$ are both the usual sample mean estimator, so by linearity of expectation, ensuring that $\mathbb{E}[h_1(X)] = \mathbb{E}[h_2(X)]$ is sufficient.

Application to MCTS. Like the technique of common random numbers, antithetic variates can be applied to UCT by modifying the way simulation trajectories are sampled. Whenever a node representing $(s_i, a_i) \in \mathcal{S} \times \mathcal{A}$ is visited during the backup phase of UCT, the realized trajectory $s_{i+1}, r_{i+1}, a_{i+1}, \ldots, s_n, r_n$ from (s_i, a_i) is now stored in memory if $T^m_{s_i,a_i} \bmod 2 = 0$. The next time this node is visited ($T^m_{s_i,a_i} \bmod 2 = 1$) during the selection phase, the previous trajectory is used to predetermine one or more antithetic events that will (partially) drive subsequent state transitions for the current simulation trajectory. After this, the memory used to store the previous simulation trajectory is released. This process alternates between the *normal simulation* and *antithetically-mapped simulation* of trajectories following action a_i.

Application to Test Domains. To apply antithetic variables, we need to describe how the antithetic events are constructed from previous simulation trajectories. In Pig, a negative correlation between the returns of pairs of simulation trajectories can be induced by forcing the roll outcomes in the second trajectory to oppose those occurring in the first trajectory. Exploiting the property that the relative worth of each pair of dice outcomes is independent

of state, a list of antithetic roll outcomes can be constructed by mapping each individual roll outcome in the first trajectory to its antithetic partner. For example, a lucky roll of ⚅⚅ was paired with the unlucky roll of ⚀⚀. Table 7.1 shows the full mapping.

Roll	Pair	Roll	Pair	Roll	Pair	Roll	Pair	Roll	Pair	Roll	Pair
⚀⚀	⚅⚅	⚁⚁	⚄⚄	⚂⚂	⚃⚃	⚀⚁	⚄⚅	⚀⚄	⚁⚅	⚁⚃	⚂⚄
⚀⚂	⚃⚅	⚀⚅	⚃⚄	⚂⚃	⚂⚃	⚀⚃	⚂⚅	⚁⚂	⚂⚄		

Table 7.1: Antithetic pairs of dice rolls for Pig.

When constructing such a mapping, it is important that outcomes be matched by probability of occurrence. To see why this is true, imagine a random variable X whose domain is $\{100, 200, 80\}$ and whose distribution is $\mathbb{P}(X = 100) = \mathbb{P}(X = 200) = \frac{1}{4}, \mathbb{P}(X = 80) = \frac{1}{2}$. So, $\mathbb{E}[X] = \frac{1}{2}(100 + 200) + \frac{1}{4}(80)$. Suppose we draw a sample $\mathbf{X}$. An antithetic sample $\mathbf{X}'$ can be constructed by replacing every instance of 100 with 200 and vice versa. The sample mean estimator of this new sample will be unbiased since $\mathbf{X}'$ is obtained by a simple relabeling of event values with that have the same probability. If two outcomes were mapped with unequal probability of occurrence, then it may be possible that $\mathbb{E}[X_i'] \neq \mathbb{E}[X_i]$ and so the combined estimator may be biased.

As with common random numbers, since all chance events all have the same form in Pig, we found that generating batches at the root worked better than recursively storing a single antithetic outcome per every state-action pair. That is, we generate a batch of future stochastic outcomes at the root for the normal simulation of (s, a) and then during the antithetically-mapped simulation of (s, a), the previous batch is converted to an antithetic batch by applying the antithetic mapping to each outcome in batch. These batches then determine the effects of chance throughout a simulation.

In Can't Stop, however, the situation is more complicated, since the relative worth of each chance event varies from state to state (*e.g.,* rolling ⚀⚀⚀⚀ is a good roll for a player with a turn progress marker on the 3 column, but a bad roll for a player whose 3 markers are already allocated to other columns). The solution was to develop a state-dependent heuristic ranking function, which would assign an index between 0 and 1295 to the 6^4 distinct chance events for a given state. Chance events that are favorable in the current state are assigned low indexes, while unfavorable events are assigned high index values. During a normal simulation, the ranking index i for each chance event is recorded. During the antithetically mapped simulation trajectory, the previously recorded rank indices are used to compute the

relevant antithetic event for the current state (*e.g.,* using $i' = 1295 - i$). This approach can be applied in a wide variety of domains where the stochastic outcomes can be ordered by how "lucky" they are e.g., suppliers' price fluctuations, rare catastrophic events, or higher than average click-through-rates.

For Dominion, a number of antithetic mappings were tried, but none provided any substantial reduction in variance. The complexity of how cards can be played to draw more cards from one's deck makes a good or bad shuffle intricately dependent on the exact composition of cards in one's deck, of which there are intractably many possibilities with no obvious symmetries. So we do not explore antithetic variates further in Dominion. This demonstrates one problem with antithetic variates: for complex domains, an antithetic pairing may be difficult to find.

7.4 Empirical Evaluation in MCTS

Each variance reduction technique is evaluated in combination with the UCT algorithm, with varying levels of search effort. In Pig, the default (rollout) policy plays the roll and stop actions with probability 0.8 and 0.2 respectively. In Can't Stop, the default policy will end the turn if a column has just been finished, otherwise it will choose to re-roll with probability 0.85. In Dominion, the default policy incorporates some simple domain knowledge that favors obtaining higher cost cards and avoiding redundant actions. The UCB constant c in Equation 2.22 was set to 100.0 for both Pig and Dominion and 5500.0 for Can't Stop. These values of c are related to the range of the utility function; specifically, they are half of the utility function's range.

For control variates, we use a mixture of online and offline estimation to determine the values of $c_{s,a}$ to use in Equation 7.2. When $T^m_{s,a} \geq 50$, the online estimate

$$-\widehat{\mathbb{C}\text{ov}}[X_{s,a}, Y_{s,a}]/\widehat{\mathbb{V}\text{ar}}[Y_{s,a}]$$

was used. If $T^m_{s,a} < 50$, the constants 6.0, 6.0 and -0.7 are used for Pig, Can't Stop and Dominion respectively. These constants were obtained by computing offline estimates of $-\widehat{\mathbb{C}\text{ov}}[X_{s,a}, Y_{s,a}]/\widehat{\mathbb{V}\text{ar}}[Y_{s,a}]$ across a representative sample of game situations. This combination gives better performance than either scheme in isolation.

Three sets of experiments were performed. The first two are used to gain a deeper understanding of the role of bias and variance in UCT. The final set of results is used to assess the overall performance of UCT when augmented with our variance reduction techniques.

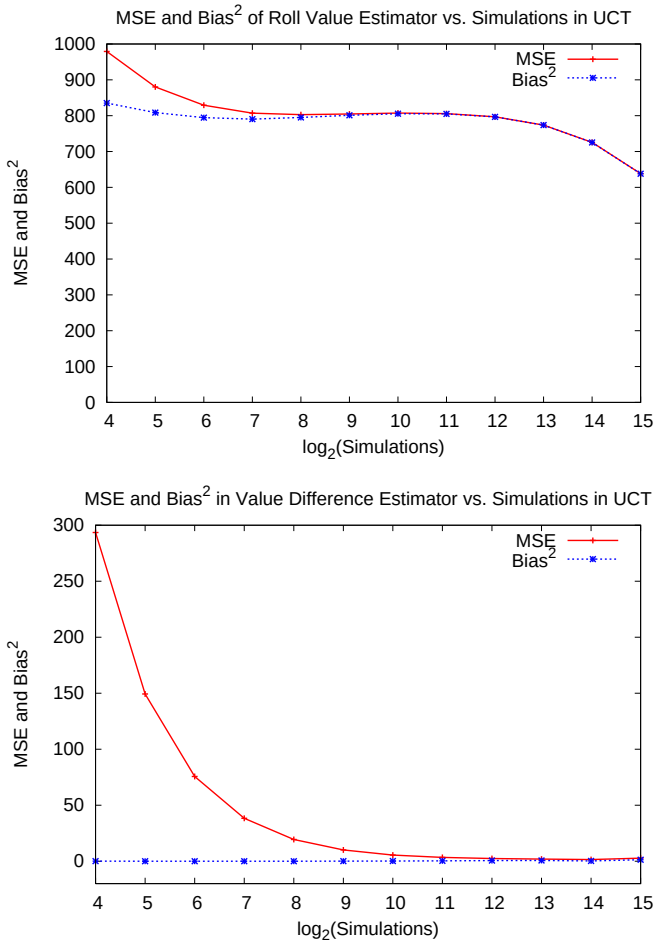


Figure 7.1: The estimated variance of the value estimates for the Roll action and estimated differences between actions on turn 1 in Pig.

Recall from Section 6.4, when assessing the quality of an estimator using *mean squared error* (MSE), it is well known that the estimation error can be decomposed into two terms, bias and variance. Therefore, when assessing the potential impact of variance reduction, it is important to know just how much of the estimation error is caused by variance as opposed to bias. Since the game of Pig has $\approx 2.4 \times 10^6$ states, we can solve it exactly offline using Expectimax Search. This allows us to compute the exact expected return $\mathbb{E}[X_{s_1} \mid \pi^*]$ of the optimal action (roll) at the starting state s_1. We use this value to compute both the bias-squared and variance component of the MSE for the estimated return of the roll action at s_1 when using UCT without variance reduction. This is shown in the leftmost graph of Figure 7.1. It seems that the dominating term in the MSE is the bias-squared. This is misleading however, since the absolute error is not the only factor in determining which action is selected by UCT. More importantly, the *difference* between the estimated returns

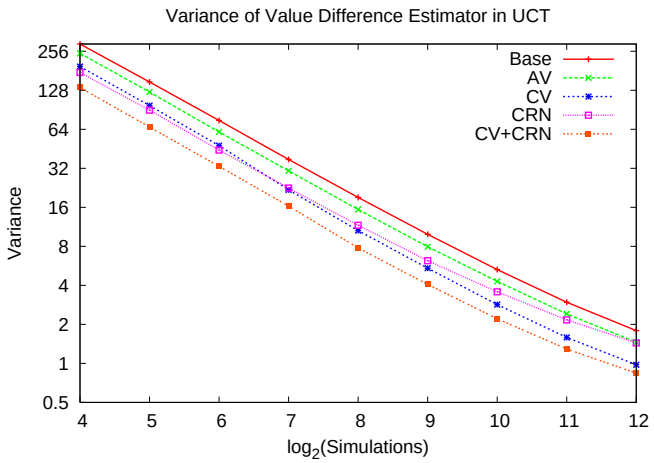


Figure 7.2: Variance of the estimator of the difference between the returns for the roll and stop actions at turn 1 in Pig. Note: both axes use a log scale.

for each action is how UCT ultimately ends up choosing its action. As Pig has just two actions, we can also compute the MSE of the estimated difference in return between rolling and stopping using UCT without variance reduction. This is shown by the bottom graph in Figure 7.1. Here we see that variance is the dominating component (the bias is within ± 2) when the number of simulations is less than 1024. The role of bias and variance will, of course, vary from domain to domain, but this result suggests that variance reduction may play an important role when trying to determine the best action.

In addition, we investigate how effective our variance reduction techniques are at reducing the variance between the estimated difference in returns of the roll and stop actions in Pig on the first turn. We tested all three of the methods, as well as a hybrid method that used both control variates and common random numbers. Figure 7.2 shows the variance of each method, as the number of simulations increase, on a log-log scale. All of the techniques lead to a substantial decrease in variance. In particular, the best method achieved lower variance than the standard estimator even when using half as many simulations.

The technique of common random numbers was quite helpful, which makes sense as it is designed specifically for decreasing the variance of differences between estimators. Furthermore, the combination of control variates with common random numbers was complementary, with both seemingly removing independent portions of the variance. Interestingly, as opposed to control variates and antithetic variates, the effectiveness of common random numbers appears to degrade as the number of simulations is increased. We believe this to be caused by the non-uniform behavior of the UCB policy, which increasingly skews the

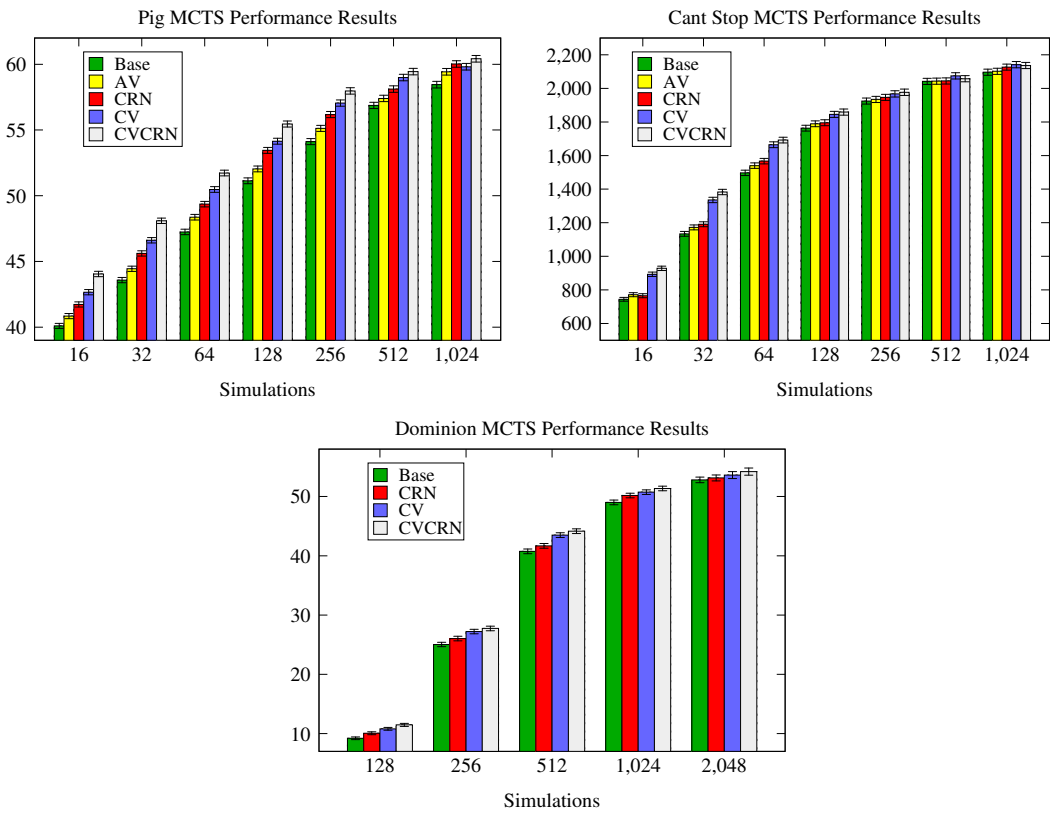


Figure 7.3: Performance Results for Pig, Can't Stop, and Dominion with 95% confidence intervals shown. Values on the vertical axis of each graph represent the average score.

simulations toward the actions with higher return estimates. This has the effect of reducing how often common events can be shared across all of the actions for a particular state.

The next set of results reports the relative performance of the variance-reduced MCTS algorithms. Figure 7.3 shows the results of our variance reduction methods on Pig, Can't Stop and Dominion. Each data point for Pig, Can't Stop and Dominion is obtained by averaging the scores obtained across 50000, 10000 and 10000 games, respectively. Such a large number of games is needed to obtain statistically significant results due to the highly stochastic nature of each domain. 95% confidence intervals are shown for each data point. In Pig, the best combination of variance reduction techniques consistently outperforms the base version of UCT, even when given twice the number of simulations. In Can't Stop, the best combination of variance reduction techniques gave a performance increase roughly equivalent to using base UCT with 50-60% more simulations. The results also show a clear benefit to using variance reduction techniques in the challenging game of Dominion. Here the best combination of variance reduction techniques leads to an improvement roughly

equivalent to using 25-40% more simulations. The use of antithetic variates in both Pig and Can't Stop gave a measurable increase in performance, however the technique was less effective than either control variates or common random numbers. Control variates was particularly helpful across all domains, and even more effective when combined with common random numbers.

Although our UCT modifications are designed to be lightweight, some additional overhead is unavoidable. Common random numbers and antithetic variates increase the space complexity of UCT by a multiplicative constant. Control variates typically increase the time complexity of each value backup by a constant. These factors need to be taken into consideration when evaluating the benefits of variance reduction for a particular domain. Note that surprising results are possible; for example, if generating the underlying chance events is expensive, using common random numbers or antithetic variates can even *reduce* the computational cost of each simulation. Ultimately, the effectiveness of variance reduction in MCTS is both domain and implementation specific. That said, we would expect our techniques to be useful in many situations, especially in noisy domains or if each simulation is computationally expensive. In our experiments, the overhead of every technique was dominated by the cost of simulating to the end of the game.

7.5 Application to MCCFR

In this section, we describe two straight-forward applications of variance reduction in MCCFR: one based on antithetic variates, and the other on common random numbers. Both are hybrids between Vanilla CFR and chance-sampling CFR.

Both applications are applied to Bluff(1,1). Recall the game of Bluff(1,1); the chance nodes can be modeled in many different ways. Two ways to model Bluff(1,1) are shown in Figure 7.4. For the purposes of this section, we assume that Bluff(1,1) is modeled using the 7 chance-node model. In this model, we can classify the chance nodes as *belonging to a player* i if the outcome of the chance node determines which information set a history in its subtree is contained. In particular, one chance node belongs to player 1 and six chance nodes belong to player 2. There is an identical model where one chance node belongs to player 2 and six chance nodes belong to player 1.

Recall that Vanilla CFR iterates over and recursively traverses the subtree under every outcome at chance nodes. In chance-sampled CFR, a single outcome is sampled at each chance node and only the one subtree under the sampled outcome is traversed.

Paired chance-sampling samples two outcomes at chance nodes. The outcomes are paired antithetically like in Pig. In Bluff(1,1), we use the mapping $((\boxdot, \star), (\boxdot, \boxdot), (\boxdot, \boxdot))$. The justification for this choice is simple: the counterfactual value is a weighted value that depends on π^{σ}_{-i} (which includes chance's probabilities), and the chance outcome may affect a player's position and hence their expected value when using σ_i.

Common chance-sampling, like Vanilla CFR, uses domain knowledge to take advantage of redundancy in the game model. Consider a hybrid version of Vanilla CFR and chance-

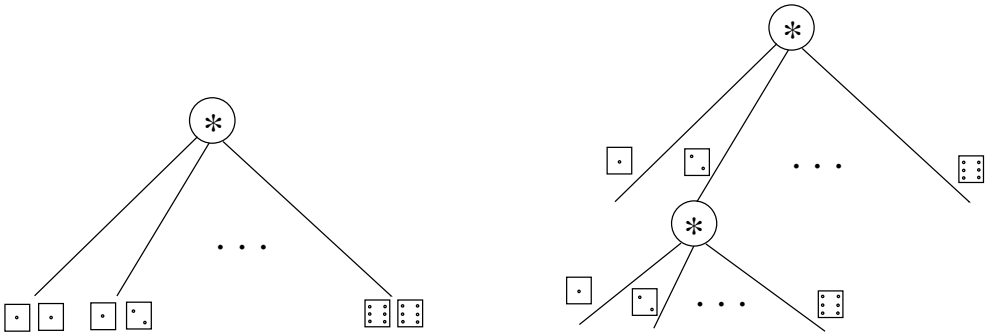


Figure 7.4: Two different ways to model how chance events occur in Bluff(1,1). The model on the left has a single chance node with 36 outcomes. The model on the right has 7 chance nodes, each with 6 outcomes.

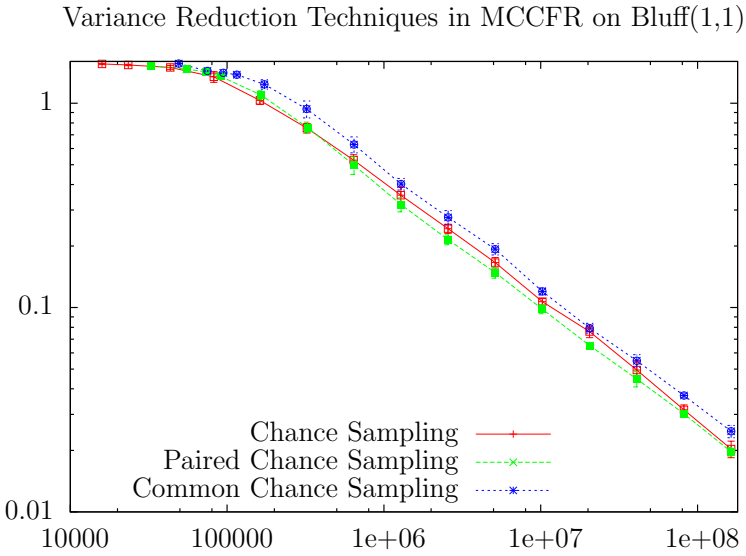


Figure 7.5: The effects of variance reduction techniques applied to chance sampling in MCCFR. As with most graphs from Chapter 4, the horizontal axis represents nodes touched and the vertical axis represents exploitability ϵ_{σ}. Each line is an average over 5 runs with error bars representing 95% confidence intervals.

sampled CFR, which iterates over every chance outcome at chance nodes belonging to i, and samples and chance nodes belonging to $-i$ and public chance nodes (a chance node *belongs* to a player if the outcome of the chance event is only revealed to that player, while a public chance node is one where all players see the outcome of the chance event). When reaching a chance node belonging to player 2, different outcomes may be sampled for each outcome of player 1, when clearly the outcome of chance nodes for player 2 represent the same strategic situation. Therefore, in common-chance sampling there is one important modification: if a chance node belonging to $-i$ is visited due to a different combination of outcomes assigned to i, the same outcome that was sampled previously is re-used. For example, in Bluff(1,1), and $i = 1$, then a single outcome is sampled and shared across each of the six chance nodes belonging to player 2.

Both paired chance-sampling and common chance-sampling were compared to chance-sampled CFR. The results are shown in Figure 7.5. We see that reducing the variance using paired chance-sampling has a small but still statistically significant effect on the convergence rate. We expect this benefit to be larger in games with more stochasticity. In contrast, common chance-sampling converges slightly slower than chance sampling. While the variance is being reduced, the benefit of sampling is likely being lost due to traversing all of the subtrees under outcomes at chance nodes belonging to i.

7.6 Chapter Summary and Conclusion

In this chapter, we have have described how to apply variance reduction techniques to Monte Carlo tree search and to MCCFR. Common random numbers and are particularly easy to implement and appear to be widely applicable. We have shown that antithetic variates can be applied via a remapping of chance node outcomes whenever a negative correlation in utility can described by this mapping. A simple application of antithetic variates also improved the performance of MCCFR in Bluff(1,1). Control variates require an additional expectation to be known, but the specific construction we present are practical since the expectations are always zero. In our experiments, the combination of control variates and common random numbers seemed to perform best in MCTS, and can result in up to an effective doubling of the number of simulations for some games.

Chapter 8

Conclusion

In this thesis, we investigated the problem of decision-making in large, two-player, zero-sum extensive-form games with imperfect information and perfect information. In particular, we focus on equilibrium approximation and decision-making in large sequential games.

We introduced Monte Carlo Counterfactual Regret Minimization (MCCFR), analyzed its general theoretical guarantees as well as the properties of two specific sampling schemes: outcome sampling and external sampling. We proposed a new theory that shows how counterfactual regret minimization can be applied to abstract games with imperfect recall, allowing us to compute approximate equilibria for much larger games. Then, we presented Monte Carlo *-Minimax Search, an asymptotically consistent sampling version of classic expectimax and *-Minimax. Lastly we showed how variance reduction techniques from the literature can be applied in the tree search setting. In all of these cases, the algorithms are thoroughly evaluated in practice by presenting the results of experiments on a number of domains.

What have we learned?

Monte Carlo Counterfactual Regret Minimization is a general family of sample-based algorithms for minimizing counterfactual regret and computing approximate equilibria. Its performance depends on the sampling scheme chosen and the structure of the game it is run on. In theory, external sampling has a better bound than outcome sampling due to the lack of the $\frac{1}{\delta}$ term. If computation time is the measure of performance, and we fix a tolerance probability p, then external sampling will provide a better convergence rate; this is because the number of iterations required by Vanilla CFR and external sampling are comparable, but iterations of external sample take roughly the root of the time taken by Vanilla CFR. The performance of MCCFR in practice is highly game-dependent, with external sampling

seeming to be the best overall choice.

When solving games using abstractions based on imperfect recall transformations, care must be taken in how the abstract games are defined. Using the most strict definition of well-formed games will ensure that CFR will also minimize regret in its perfect recall refinement, at the same rate. A penalty in the bound is paid in skew well-formed games. Relaxing the conditions of these definitions without losing convergence properties seems difficult. The convergence rates of CFR on these abstract games in practice are rather mixed. Even when a small payoff skew is introduced, the convergence rate can suffer quite heavily. And yet, even when the game is not well-formed, memory can be saved with comparatively little effect on the convergence rate. One possibility is that satisfying some conditions is more important than others, but more investigation is required to make any further claims.

In stochastic games with perfect information, Monte Carlo *-Minimax Search provides an alternative to the popular Monte-Carlo Tree Search. In theory, the estimates computed by MCMS converge to the true Minimax values. In practice, we notice that MCMS can reach lower depths in the same amount of time as *-Minimax, at the cost of increased variance but typically at a higher accuracy (less bias and mean squared error). In head-to-head matches in small to medium-sized densely stochastic games, MCMS is shown to compete with the state of the art (MCTS with double-progressive widening). In a larger densely stochastic game (Can't Stop), MCMS beats both *-minimax and the state of the art convincingly.

Finally, we show that applying variance reduction techniques in the context of tree search and MCCFR can be helpful. Common random numbers and are particularly easy to implement and seem widely applicable. Antithetic variates require some careful construction but are also rather straight-forward. Control variates require an additional expectation to be known, but the trajectory-based form presented makes this more practical. In practice, the combination of control variates and common random numbers can increase performance of MCTS to a level equivalent to a doubling in the number of simulations.

8.1 Future Work

Here, we list a number of potential future research questions based on this work.

1. **Pure CFR.** Pure CFR is an idea originally proposed to the University of Alberta Computer Poker Research Group by Oskari Tammelin, a hobbyist programmer interested in MCCFR. In Pure CFR, deterministic strategies τ_1^t and τ_2^t are sampled from the σ_1^t and σ_2^t. Then, Vanilla or chance-sampled CFR computes regret assuming both

players play these sampled deterministic strategies. Pure CFR seems like another MCCFR sampling scheme, and so it may be possible to derive theoretical properties from the general MCCFR theorem as was done for outcome and external sampling. From an implementation standpoint: since many of the $\sigma(I, a)$ are zero, the CFR algorithm can prune many sub-trees. Also, if the payoffs are expressed as integers, the products π_i, π_{-i} and values v_i will always be integer values, leading to faster operations and less memory overhead.

2. **Averaging in Imperfect Recall Abstractions.** The full averaging scheme used to show the convergence rates of the imperfect recall abstract game requires memory linear in the size of the full game. In Section 5.4, we show that in practice using the abstract averaging seems show a very small loss in convergence rate. However, there is no theoretical analysis to explain this or to answer when it may be true. This is important, as the most recent Poker AI competition bots tend to apply CFR to imperfect recall abstractions without averaging in the full strategy space.

3. **MCMS Follow-up.** While we have shown that the performance of MCMS is competitive with the state of the art in Can't Stop, we ultimately hope to show that it can perform well in larger games such. We hope to obtain a detailed analysis of the effect of the width parameter in MCMS, including dynamically-determined widths based on depth. In addition, we are interested in whether the bound from Theorem 8 can be used to inspire specific values of the sample width c to use in practice. Finally, we hope to derive a convergence bound for the case of sampling without replacement.

4. **Variance Reduction.** Variance reduction may be a practical tool to enhance modern Monte Carlo search based AI in adversarial domains. We have shown that the techniques can be practical in two-player games in MCCFR, and the extension to two-player MCTS seems natural. Also, since MCMS introduces variance, it also seems like an appropriate setting to apply these techniques. Other techniques, such as stratified sampling, did not work very well in our unreported exploratory experiments; it would be nice to know why this is, or to discover the circumstances under which search can benefit from stratified sampling.

5. **Full Bluff AI and FSICFR Follow-up.** Neller and Hnath's FSICFR [77] is a more efficient form of chance-sampled CFR specifically designed for Bluff-like games. There is no formal analysis or proof of this claim. Also, their paper does not show

FSICFR converging to an equilibrium in practice by minimizing exploitability, not even for Bluff(1,1). An interesting research direction would be to formally prove the equivalence of their algorithm and chance-sampled CFR as well as analyzing FSICFR's complexity. An interesting follow-up would be to solve larger abstractions, and create an AI for Bluff(5,5), the version of the games played by humans. Defeating expert-level humans at this game would make it the largest imperfect information game for such a feat: excluding all of its subgames, Bluff(5,5) has $252 \cdot 2^{60} \approx 2.9 \cdot 10^{20}$ information sets compared to Two-Player Limit Texas Hold'em, which has an estimated $3.2 \cdot 10^{14}$ information sets.

Bibliography

[1] I. Adler, N. Karmarkar, M.G.C. Resende, and G. Veiga. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44:297–335, 1989.

[2] V. Allis. A knowledge-based approach to Connect Four. Master's thesis, Vrije Universiteit, October 1988.

[3] V. Allis. Proof-number search. *Artificial Intelligence*, 66(1):91–124, March 1994.

[4] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50:5–43, 2003.

[5] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, pages 322–331, 1995.

[6] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2/3):235–256, 2002.

[7] B. W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3):327–350, 1983.

[8] L. Barone and L. While. Evolving adaptive play for simplified Poker. In *Proceedings of IEEE International Conference on Computational Intelligence (ICEC-98)*, 1998.

[9] D. P. Bertsekas and D. A. Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 1999.

[10] D. Billings. Computer Poker. Master's thesis, University of Alberta, August 1995.

[11] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale Poker. In *Proceedings of the 2003 International Joint Conference on Artificial Intelligence IJCAI-2003*, 2003.

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

[13] A. Blum and Y. Mansour. Learning, regret minimization, and equilibria. In *Algorithmic Game Theory*, chapter 4. Cambridge University Press, 2007.

[14] C.B. Browne, E. Powley, D. Whitehouse S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[15] M. Buro. The Othello match of the year: Takeshi Murakami vs. Logistello. *International Computer Chess Association (ICCA)*, 20(3):189–193, 1997.

[16] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus. An adaptive sampling algorithm for solving Markov Decision Processes. *Operations Research*, 53(1):126–139, January 2005.

[17] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-Carlo tree search: A new framework for game AI. In Michael Mateas and Chris Darken, editors, *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 216–217. AAAI Press, Menlo Park, CA., 2008.

[18] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-Carlo tree search: A new framework for game AI. In Michael Mateas and Chris Darken, editors, *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 216–217. AAAI Press, Menlo Park, CA., 2008.

[19] G. Chaslot, M. Winands, H. J. van den Herik, J. Uiterwijk, and B. Bouzy. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(3):343–357, 2008.

[20] A. Couetoux, J-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In *LION'11: Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, Italy, January 2011.

[21] R. Coulom. Computing ELO ratings of move patterns in the game of Go. *International Computer Games Association*, 30(4):198–208, 2007.

[22] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th international conference on Computers and games*, CG'06, pages 72–83, Berlin, Heidelberg, 2007. Springer-Verlag.

[23] P. I. Cowling, E. J. Powley, and D. Whitehouse. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, December 2012.

[24] P. I. Cowling, C. D. Ward, and E. J. Powley. Ensemble determinization in Monte Carlo tree search for the imperfect information card game Magic: The Gathering. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):241–257, December 2012.

[25] J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[26] R. Eckhard. Stan Ulam, John von Neumann and the Monte Carlo method. *Los Alamos Science*, 15:131–136, 1987.

[27] H. Finnsson and Y. Björnsson. Simulation-based approach to general game-playing. In *The Twenty-Third AAAI Conference on Artificial Intelligence*, pages 259–264. AAAI Press, 2008.

[28] Y. Freund and R. E. Shapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Second European Conference (EuroCOLT'95)*, pages 23–37. Springer-Verlag, 1995.

[29] R. Gasser. Solving Nine Men's Morris. *Computational Intelligence*, 12:24–41, 1996.

[30] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, March 2012.

[31] S. Gelly and Y. Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *NIPS 2006 Workshop on Online Trading Between Exploration and Exploitation*, 2006.

[32] M. Genesereth and N. Love. General game-playing: Overview of the AAAI competition. *AI Magazine*, 26:62–72, 2005.

[33] R. Gibson, M. Lanctot, N. Burch, D. Szafron, and M. Bowling. Generalized sampling and variance in counterfactual regret minimization. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, 2012.

[34] Richard Gibson, Neil Burch, Marc Lanctot, and Duane Szafron. Efficient Monte Carlo counterfactual regret minimization in games with many player actions. In *Advances in Neural Information Processing Systems 25*, 2012. To appear.

[35] A. Gilpin. *Algorithms for Abstracting and Solving Imperfect Information Games*. PhD thesis, Carnegie Mellon University, 2009.

[36] A. Gilpin, S. Hoda, J. Peña, and T. Sandholm. Gradient-based algorithms for finding nash equilibria in extensive form games. In *3rd International Workshop on Internet and Network Economics (WINE'07)*, 2007.

[37] M. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14:303–358, 2001.

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

[39] G. Gordon. One card Poker. `http://www.cs.cmu.edu/~ggordon/poker/`.

[40] G. Gordon. No-regret algorithms for online convex programs. In *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems (NIPS-2006)*, 2006.

[41] G. J. Gordon. No-regret algorithms for structured prediction problems. Technical Report CMU-CALD-05-112, Carnegie Mellon University, 2005.

[42] A. Greenwald, Z. Li, and C. Marks. Bounds for regret-matching algorithms. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2005.

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

[44] T. Hauk, M. Buro, and J. Schaeffer. Rediscovering *-minimax search. In *Proceedings of the 4th international conference on Computers and Games*, CG'04, pages 35–50, Berlin, Heidelberg, 2006. Springer-Verlag.

[45] C. Heyden. Implementing a computer player for Carcassonnne. Master's thesis, Department of Knowledge Engineering, Maastricht University, 2009.

[46] S. Hoda, A. Gilpin, and J. Pe na. A gradient-based approach for computing Nash equilibria of large sequential games. *Optimization Online*, July 2007. `http://www.optimization-online.org/DB_HTML/2007/07/1719.html`.

[47] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[48] F. Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Championship*. Princeton University Press, 2006.

[49] R. Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley & Sons, 1965.

[50] E. Jackson. Slumbot: An implementation of counterfactual regret minimization on commodity hardware. In *Proceedings of AAAI 2012 Poker Symposium*, 2012.

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

[52] M. Johanson, N. Bard, M. Lanctot, R. Gibson, and M. Bowling. Efficient Nash equilibrium approximation through Monte Carlo counterfactual regret minimization. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012.

[53] M. Johanson, K. Waugh, M. Bowling, and M. Zinkevich. Accelerating best response calculation in large extensive games. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[54] M. Johanson, M. Zinkevich, and M. Bowling. Computing robust counter-strategies. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1128–1135, 2008. A longer version is available as a University of Alberta Technical Report, TR07-15.

[55] Michael Johanson and Richard Gibson. Personal communication, 2012.

[56] M. Kaneko and J. J. Kline. Behavior strategies, mixed strategies and perfect recall. *International Journal of Game Theory*, 4:127–145, 1995.

[57] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[58] M. J. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov Decision Processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1324–1331, 1999.

[59] G. Kendall and M. Willdig. An investigation of an adaptive Poker player. In *Proceedings of 14th Australian Joint Conference on Artificial Intelligence*, 2001.

[60] R. Knizia. *Dice Games Properly Explained*. Blue Terrier Press, 2010.

[61] L. Kocsis and C. Szepesvári. Bandit-based Monte Carlo planning. In *15th European Conference on Machine Learning*, pages 282–293, 2006.

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

[63] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC '94)*, pages 750–759, 1994.

[64] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94:167–215, 1997.

[65] H. W. Kuhn. Simplified two-person Poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

[66] H. W. Kuhn. Extensive games and the problem of information. *Contributions to the Theory of Games*, 2:193–216, 1953.

[67] M. Lanctot, R. Gibson, N. Burch, and M. Bowling. No-regret learning in extensive-form games with imperfect recall. In *Proceedings of the Twenty-Ninth International Conference on Machine Learning (ICML 2012)*, 2012.

[68] M. Lanctot, R. Gibson, N. Burch, M. Zinkevich, and M. H. Bowling. No-regret learning in extensive-form games with imperfect recall. *CoRR*, abs/1205.0622, 2012.

[69] M. Lanctot, A. Saffidine, J. Veness, and C. Archibald. Sparse sampling for adversarial games. In *Proceedings of the ECAI Computer Games Workshop*, pages 37–49, 2012.

[70] M. Lanctot, K. Waugh, M. Bowling, and M. Zinkevich. Sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems (NIPS 2009)*, 2009.

[71] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling. Monte Carlo sampling for regret minimization in extensive games. Technical Report TR09-15, University of Alberta, 2009. http://www.cs.ualberta.ca/research/techreports/2009/TR09-15.php.

[72] J. Long. *Combining Search and Inference to Play World-Caliber Skat*. PhD thesis, University of Alberta, 2011.

[73] J. Long, N.R. Sturtevant, M. Buro, and T. Furtak. Understanding the success of perfect information monte carlo sampling in game tree search. *AAAI Conference on Artificial Intelligence*, pages 134–140, 2010.

[74] R. J. Lorentz. An MCTS program to play Einstein Würfelt Nicht! In *Proceedings of the 12th International Conference on Advances in Computer Games*, 2011.

[75] M.K. Warmuth N. Littlestone. The weighted majority algorithm. In *30th Annual Symposium on Foundations of Computer Science (SCFS 1989)*, pages 256–261, 1989.

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

[77] T. W. Neller and S. Hnath. Approximating optimal Dudo play with fixed-strategy iteration counterfactual regret minimization. In *Computers and Games*, 2011.

[78] T. W. Neller and C.G.M. Pressor. Optimal play of the dice game Pig. *Undergraduate Mathematics and Its Applications*, 25(1):25–47, 2004.

[79] B. L. Nelson. Control variate remedies. *Operations Research*, 38(6):pp. 974–992, 1990.

[80] P. Nijssen and M. H. M. Winands. Monte carlo tree search for the hide-and-seek game scotland yard. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):282–294, December 2012.

[81] M.J. Osborne. *An Introduction to Game Theory*. Oxford University Press, 2004.

[82] D. Papp. Dealing with imperfect information in Poker. Master's thesis, University of Alberta, November 1998.

[83] M. Piccione and A. Rubinstein. On the interpretation of decision problems with imperfect recall. In *Proceedings of the 6th Conference on THeoretical Aspects of Rationality and Knowledge*, pages 75–76. Morgan Kaufmann Publishers Inc., 1996.

[84] Annual Computer Poker Competition Organizing Committee. Annual computer Poker competition, 2012. http://www.computerpokercompetition.org/.

[85] AAAI Computer Poker Organizing Committee. The first computer Poker competition, 2006. http://www.aaai.org/Conferences/AAAI/2006/aaai06poker.php.

[86] AAAI Computer Poker Organizing Committee. The first man vs. machine Poker championship, 2007. http://webdocs.cs.ualberta.ca/~games/poker/man-machine/2007/.

[87] Computer Poker Research Group. The second man vs. machine Poker championship, 2008. http://webdocs.cs.ualberta.ca/~games/poker/man-machine/.

[88] M. Ponsen, S. de Jong, and M. Lanctot. Computing approximate Nash equilibria and robust best-responses using sampling. *Journal of Artificial Intelligence Research*, 42:575–605, 2011.

[89] E.J. Powley, D. Whitehouse, and P.I. Cowling. Determinization and information set Monte Carlo tree search for the card game Dou Di Zhu. In *IEEE Conference on Computational Intelligence in Games (CIG 2011)*, 2012.

[90] R. Ramanujan and B. Selman. Trade-offs in sampling-based adversarial planning. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 2011.

[91] Carnegie Mellon Media Relations. Carnegie mellon computer Poker program sets its own Texas Hold'em strategy, July 2006.

[92] J. W. Romein and H. E. Bal. Solving Awari with parallel retrograde analysis. *Computer*, 36(10):26–33, 2003.

[93] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[94] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 3rd edition, 2010.

[95] S. Sackson. Can't Stop. *Ravensburger*, 1980.

[96] A. Samuel. Some studies in machine learning using the game of Checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

[97] J. Scarne. Scarne on dice. *Harrisburg, PA: Military Service Publishing Co*, 1945.

[98] J. Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, 2nd edition, 2009.

[99] J. Schaeffer, D. Billings, M. de L. P. Castillo, and D. Szafron. Learning to play strong Poker. In *International Conference on Machine Learning 1999 Workshop Machine Learning in Game Playing (ICML)*, 1999.

[100] J. Schaeffer, R. Lake, P. Lu, and M. Bryant. Chinook: The world man-machine Checkers champion. *AI Magazine*, 17(1):21–29, 1996.

[101] J. Schäfer. The UCT algorithm applied to games with imperfect information. Master's thesis, Otto-von-Guericke-Universität Magdeburg, 2008.

[102] T. Schauenberg. Opponent modelling and search in Poker. Master's thesis, University of Alberta, Spring 2006.

[103] M. Shafei, N. Sturtevant, and J. Schaeffer. Comparing UCT versus CFR in simultaneous games. In *Proceeding of the IJCAI Workshop on General Game-Playing (GIGA)*, 2009.

[104] C. Shannon. A chess-playing machine. *Scientific American*, pages 48–51, 1950.

[105] J. Shi and M. Littman. Abstraction models for game-theoretic Poker. In *Computers and Games*, Lecture Notes in Computer Science, pages 333–345. Springer-Verlag, 2001.

[106] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[107] N. Sturtevant. An analysis of UCT in multi-player games. *International Computer Games Journal*, 31(4):195–208, 2008.

[108] D. X. Vaccarino. Dominion. *Rio Grande Games*, 2008.

[109] E.C.D. van der Werf, H.J. van den Herik, and J.W.H.M. Uiterwijk. Solving Go on small boards. *International Computer Games Association Journal*, 26(2):92–107, 2003.

[110] J. Veness, M. Lanctot, and M. Bowling. Variance reduction in Monte-Carlo tree search. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1836–1844. 2011.

[111] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 1947.

[112] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 2nd edition, 1947.

[113] B. von Stengel. Equilibrium computation for two-player games in strategic and extensive form. In *Algorithmic Game Theory*, chapter 4. Cambridge University Press, 2007.

[114] AAAI Man vs. Machine Poker Organizing Committee. The first man vs. machine Poker championship, 2007. `http://www.poker-academy.com/man-machine/results.php`.

[115] T. J. Walsh, S. Goschin, and M. L. Littman. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings of The Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, 2010.

[116] L. Wasserman. *All of Statistics*. Springer Science+Business Media Inc., 2004.

[117] K. Waugh, D. Schnizlein, M. Bowling, and D. Szafron. Abstraction pathologies in extensive games. In *The Eight International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 781–788, 2009.

[118] K. Waugh, M. Zinkevich, M. Johanson, M. Kan, D. Schnizlein, and M. Bowling. A practical use of imperfect recall. In *Proceedings of SARA 2009: The Eighth Symposium on Abstraction, Reformulation and Approximation*, 2009.

[119] M. White and M. Bowling. Learning a value analysis tool for agent evaluation. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1976–1981, 2009.

[120] M. H. M. Winands and Y. Björnsson. Evaluation function based Monte-Carlo Lines of Action. In *Proceedings of the 12th International Conference on Advances in Computer Games*, ACG'09, pages 33–44, Berlin, Heidelberg, 2010. Springer-Verlag.

[121] J. Yang, S. Liao, and M. Pawlak. On a decomposition method for finding solution in Hex game. *International Conference on Application and Development of Computer Games in the 21st Century*, pages 96–111, 2001.

[122] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of Twentieth International Conference on Machine Learning (ICML-2003)*, 2003.

[123] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. Technical Report TR07-14, University of Alberta, September 2007.

[124] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS)*, 2008.

Glossary

abstract averaging A method of computing the average strategy when minimizing counterfactual regret in an abstract game of imperfect recall which involves by combining the updates from all $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$.

abstract game a strictly smaller game Γ' than some other game Γ such that the abstract game is smaller (every abstract information set is a composition of one or more information sets in the larger game) .

alternating form a particular implementation of outcome sampling where the regret is updated for actions at each information set belonging to a single player, alternating the update player on each new sample.

behavioral strategy a set of distributions over actions $A(I)$, one for each information set $I \in \mathcal{I}_i$ for player i.

best response strategy a strategy that yields the maximum payoff against a particular set of strategies used by the opponents.

block a subset of the terminal history set $Q \subseteq Z$.

chance event a history $h \in H \backslash Z$ such that $P(h) = c$.

chance event outcome a particular action $a \in A(h)$ when h is a chance event.

counterfactual regret the regret for *not* taking some action a at information set I and instead playing with some strategy $\sigma^t(I)$, weighted by the opponent's probability of reaching I, $\pi_{-i}(I)$.

determinization a method used in imperfect information search algorithms involving sampling a state from the current information set and then running a search algorithm on the perfect information game rooted at the sampled node.

epsilon-on-policy a process to ensure exploration by sampling action a at I according to distribution $\epsilon \cdot \text{Unif}(I) + (1 - \epsilon) \cdot \sigma(I, a)$.

exploitability a value $\epsilon_\sigma = \epsilon_1 + epsilon_2$ assigned to a profile σ where σ_1 is exploitable by some amount ϵ_2 and σ_2 is exploitable by some amount ϵ_1.

external regret The sum of the differences in utility of playing the single best option and utility of the actual option chosen, over all T trials/iterations.

external sampling A sampling scheme for MCCFR where no sampling is done at nodes belonging to player i while opponent and chance actions are sampled on-policy.

full averaging A method of computing the average strategy when minimizing counterfactual regret in an abstract game of imperfect recall which involves computing the full average strategy separately from the accumulated regret.

full minimax The minimax algorithm that is not depth-limited and therefore continues searching to the leaves of the game tree, returning only true payoff values (never stopping early to return heuristic evaluations).

history a particular sequence of actions taken by players; in a perfect recall game every history corresponds to a unique node in the game tree.

information set a set of histories $I \in \mathcal{I}_i$ that a player (i) cannot distinguish between due to information not known by that player.

lazy-weighted averaging a correct way compute the average strategy in MCCFR which involves storing partial previous updates in branches that were not sampled and pushing these updates down as actions are sampled.

minimax value the expected value of the utility at a state in a perfect information game, assuming that both players employ a minimax strategies in the subgames of all the optimal actions.

mixed Nash equilibrium a collection of mixed strategies, one for each player, for which each player has no incentive to deviate from unilaterally.

mixed strategy a probability distribution over pure strategies.

normal form game a game where all players choose a single action (pure strategy) to play simultaneously without knowledge of the other players' choices.

optimistic averaging a method of computing the average strategy in MCCFR that involves keeping a counter c_I of the last time I was visit ted, and performing $(c_I - t)$ updates to the average when visiting I.

outcome sampling an MCCFR sampling scheme where each block $Q \in \mathcal{Q}$ contains exactly one terminal history, i.e. $|Q| = 1$.

parallel form a particular implementation of outcome sampling where the regret is updated for each action at each information set belong to both players from a single sample.

perfect recall each player remembers the information that is revealed to them while playing and the exact order in which each piece of information is revealed to them.

policy The analogue of a strategy in perfect information games, particularly single-agent problems. A policy is a collection of probability distributions of the form $\pi(a|s)$ over actions $a \in \mathcal{A}$ at state s.

prefix history a history $h \sqsubseteq h'$ such that $h' \neq h$ and h' is a successor of h or there exists a set of intermediate histories $\{h_1, h_2, \ldots, h_n\}$ for $n \geq 1$ such that h_1 is a successor of h, h_{i+1} is a successor of h_i, and h' is a successor of h_n.

pure Nash equilibrium a collection of pure strategies, one for each player, for which each player has no incentive to deviate from unilaterally.

pure strategy a single decision in a normal form game; a collection of tuples $\{(I, a) : I \in \mathcal{I}_i, a \in A(I)\}$ for player i such that every I appears exactly once in an extensive-form game.

pure strategy profile a collection of pure strategies, one per player; in this thesis most commonly a pair of pure strategies.

regret A quantification, based on difference in utility, of the amount a regret associated decisions that were made during the course of a number of trials $1, 2, \cdots, T$ versus a set of other decisions that could have been made instead.

regret minimization an iterative process that leads to zero average external regret as the number of iterations T approaches infinity.

regret-matching A process where a new distribution is obtained by normalizing the positive portions of a regret vector, or using a uniform distribution if all entries are non-positive.

sampled counterfactual value an estimator of the counterfactual value at I given a strategy profile σ and a sampled block Q_j.

states a node in an extensive-form game tree, also called a history in games with perfect recall (both in perfect and imperfect information games).

stochastically-weighted averaging a method of updating the average strategy in MCCFR where the updates are weighted by the inverse probability of sampling the history, approximated by $\frac{1}{q(h)}$.

strictly informative a property of a game that ensures that two distinct histories, one being a prefix of the other, are never in the same information set.

terminal history a history from root to leaf.

Appendix A

Proofs

A.1 MCCFR Theorem Proofs

Lemma 8. *For all real a, define $a^+ = \max(a, 0)$. For all a, b, it is the case that*

$$\left((a+b)^+\right)^2 \leq (a^+)^2 + 2(a^+)b + b^2$$

Proof.

$$(a+b)^+ \leq (a^+ + b)^+ \leq \left|a^+ + b\right|$$

Squaring both sides gives the result. □

A.1.1 Regret-Matching

Lemma 9. *If regret-matching is used, then*

$$\sum_{a \in A} \bar{R}^{T,+}(a) r^{T+1}(a) \leq 0.$$

Proof. If $\bar{R}^{T,+}_{sum} \leq 0$, then for all $a \in A$, $\bar{R}^+_T(a) = 0$, and the result is trivial. Otherwise:

$$\begin{aligned}
\sum_{a \in A} \bar{R}^{T,+}(a) r^{T+1}(a) &= \sum_{a \in A} \bar{R}^{T,+}(a)(u^{T+1}(a) - u^{T+1}(\sigma^t)) \\
&= \left(\sum_{a \in A} \bar{R}^{T,+}(a) u^{T+1}(a)\right) - \left(u^{T+1}(\sigma_t) \sum_{a \in A} \bar{R}^{T,+}(a)\right) \\
&= \left(\sum_{a \in A} \bar{R}^{T,+}(a) u^{T+1}(a)\right) - \left(\sum_{a' \in A} \sigma^{T+1}(a') u^{T+1}(a')\right) \bar{R}^{T,+}_{sum} \\
&= \left(\sum_{a \in A} \bar{R}^{T,+}(a) u^{T+1}(a)\right) - \left(\sum_{a' \in A} \frac{\bar{R}^{T,+}(a')}{\bar{R}^{T,+}_{sum}} u^{T+1}(a')\right) \bar{R}^{T,+}_{sum} \\
&= \left(\sum_{a \in A} \bar{R}^{T,+}(a) u^{T+1}(a)\right) - \left(\sum_{a' \in A} \bar{R}^{T,+}(a') u^{T+1}(a')\right) \\
&= 0
\end{aligned}$$

□

Theorem 2 *When regret-matching is used:*

$$\sum_{a\in A}(\bar{R}^{T,+}(a))^2 \leq \frac{1}{T^2}\sum_{t=1}^{T}|A|(\Delta^t)^2.$$

Proof. We prove this by induction on T. The base case (for $T = 1$) is trivial. Assuming this holds for $T-1$, we prove it holds for T. Since $\bar{R}^T(a) = \frac{(T-1)}{T}\bar{R}^{T-1}(a) + \frac{1}{T}r^T(a)$, by Lemma 8:

$$(\bar{R}^{T,+}(a))^2 \leq \left(\frac{(T-1)\bar{R}^{T-1,+}(a)}{T}\right)^2 + 2\frac{T-1}{T^2}\bar{R}^{T-1,+}(a)r^T(a) + \left(\frac{r^T(a)}{T}\right)^2$$

Summing over all $a \in A$ gives:

$$\sum_{a\in A}(\bar{R}^{T,+}(a))^2 \leq \sum_{a\in A}\left(\left(\frac{T-1}{T}\right)^2(\bar{R}^{T,+}(a))^2 + 2\frac{T-1}{T^2}\bar{R}^{T-1,+}(a)r^T(a) + \frac{1}{T^2}(r^T(a))^2\right)$$

By Lemma 9, the middle term $\sum_{a\in A}\bar{R}^{T-1,+}(a)r^T(a) = 0$, so pushing the sum through:

$$\sum_{a\in A}(\bar{R}^{T,+}(a))^2 \leq \left(\left(\frac{T-1}{T}\right)^2\sum_{a\in A}(\bar{R}^{T-1,+}(a))^2\right) + \left(\frac{1}{T^2}\sum_{a\in A}(r^T(a))^2\right)$$

By the induction hypothesis:

$$\sum_{a\in A}(\bar{R}^{T-1,+}(a))^2 \leq \frac{1}{(T-1)^2}\sum_{t=1}^{T-1}|A|(\Delta_t)^2.$$

and since $|r^T(a)| \leq \Delta^T$:

$$\sum_{a\in A}(\bar{R}^{T,+}(a))^2 \leq \frac{1}{T^2}\left(\sum_{t=1}^{T-1}|A|(\Delta^t)^2\right) + \frac{|A|(\Delta^T)^2}{T^2} = \frac{1}{T^2}\sum_{t=1}^{T}|A|(\Delta^t)^2.$$

□

A.2 Proofs of Theorems 6 and 7 for Imperfect Recall Games

To prove Theorems 6 and 7, we need to first make an observation about regrets and prove a lemma whose result we be used in the proof. By the definition of counterfactual value (equation 2.15), the regrets between Γ and a perfect recall refinement $\breve{\Gamma}$ are additive; specifically, for $I \in \mathcal{I}_i$ in Γ,

$$R_i^T(I,a) = \sum_{\breve{I}\in\breve{\mathcal{P}}(I)} R_i^T(\breve{I},a). \tag{A.1}$$

We now provide a lemma that generalizes [124, Theorem 4] by showing that if the immediate counterfactual regrets of each $\breve{I} \in \breve{\mathcal{P}}(I)$ are proportional up to some difference D, then the average regret can be bounded above:

Lemma 10. *Let $\breve{\Gamma}$ be a perfect recall refinement of a game Γ. If for all $I \in \mathcal{I}_i$, $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$, and $a \in A(I)$, there exist constants $C_{\breve{I},\breve{I}',a}, D_{\breve{I},\breve{I}',a} \in [0,\infty)$ such that*

$$\frac{1}{T}\left|R_i^{T,+}(\breve{I},a) - C_{\breve{I},\breve{I}',a}R_i^{T,+}(\breve{I}',a)\right| \leq D_{\breve{I},\breve{I}',a}, \tag{A.2}$$

then the average regret in $\breve{\Gamma}$ is bounded by

$$\frac{\breve{R}_i^T}{T} \leq \frac{\Delta_i C\sqrt{|A_i|}}{\sqrt{T}} + \sum_{I\in\mathcal{I}} |\breve{\mathcal{P}}(I)|D_I,$$

where

$$C = \sum_{I\in\mathcal{I}_i} \max_{\breve{I},\breve{I}'\in\breve{\mathcal{P}}(I),a\in A(I)} C_{\breve{I},\breve{I}',a} \qquad \textit{and} \qquad D_I = \max_{\breve{I},\breve{I}'\in\breve{\mathcal{P}}(I),a\in A(I)} D_{\breve{I},\breve{I}',a}.$$

Proof.

$$\begin{aligned}
\breve{R}_i^T &\leq \sum_{\breve{I}\in\breve{\mathcal{I}}_i} \max_{a\in A(I)} R_i^{T,+}(\breve{I},a) \text{ by [124, Theorem 3]} \\
&= \sum_{I\in\mathcal{I}_i} \sum_{\breve{I}\in\breve{\mathcal{P}}(I)} \max_{a\in A(I)} R_i^{T,+}(\breve{I},a) \text{ by definition of a perfect recall refinement} \\
&\leq \sum_{I\in\mathcal{I}_i} |\breve{\mathcal{P}}(I)| R_i^{T,+}(\breve{I}^*,a^*) \text{ where } \breve{I}^* = \arg\max_{\breve{I}\in\breve{\mathcal{P}}(I)} \max_{a\in A(I)} R_i^{T,+}(\breve{I},a) \\
&\quad \text{and } a^* = \arg\max_{a\in A(I)} R_i^{T,+}(\breve{I}^*,a) \\
&\leq \sum_{I\in\mathcal{I}_i} |\breve{\mathcal{P}}(I)| \left(C_{\breve{I}^*,\breve{I}^{**},a^*} R_i^{T,+}(\breve{I}^{**},a^*) + TD_{\breve{I}^*,\breve{I}^{**},a^*}\right) \text{ by (A.2),} \\
&\quad \text{where } \breve{I}^{**} = \arg\min_{\breve{I}\in\breve{\mathcal{P}}(I)} R_i^T(\breve{I},a^*) \\
&\leq \sum_{I\in\mathcal{I}_i} |\breve{\mathcal{P}}(I)| C_{\breve{I}^*,\breve{I}^{**},a^*} \left(\frac{1}{|\breve{\mathcal{P}}(I)|} \sum_{\breve{I}\in\breve{\mathcal{P}}(I)} R_i^T(\breve{I},a^*)\right)^+ + T\sum_{I\in\mathcal{I}_i} |\breve{\mathcal{P}}(I)|D_I \\
&\quad \text{because the minimum is less than the average and } (\cdot)^+ \text{ is monotone increasing} \\
&= \sum_{I\in\mathcal{I}_i} C_{\breve{I}^*,\breve{I}^{**},a^*} R_i^{T,+}(I,a^*) + T\sum_{I\in\mathcal{I}_i} |\breve{\mathcal{P}}(I)|D_I \text{ by (A.1)} \\
&\leq \sum_{I\in\mathcal{I}_i} C_{\breve{I}^*,\breve{I}^{**},a^*} T\sqrt{\sum_{a\in A(I)} \left(\frac{R_i^{T,+}(I,a)}{T}\right)^2} + T\sum_{I\in\mathcal{I}_i} |\breve{\mathcal{P}}(I)|D_I \\
&\leq \sum_{I\in\mathcal{I}_i} C_{\breve{I}^*,\breve{I}^{**},a^*} \Delta_i \sqrt{|A(I)|}\sqrt{T} + T\sum_{I\in\mathcal{I}_i} |\breve{\mathcal{P}}(I)|D_I \\
&\quad \text{by [71, Theorem 6]} \\
&\leq \Delta_i C\sqrt{|A_i|}\sqrt{T} + T\sum_{I\in\mathcal{I}_i} |\breve{\mathcal{P}}(I)|D_I.
\end{aligned}$$

Dividing both sides by T establishes the lemma. □

Note that if Γ has perfect recall, then the constants $C_{I,I,a} = 1$ and $D_{I,I,a} = 0$ for all $I \in \mathcal{I}_i$ and $a \in A(I)$ satisfy the condition of Lemma 10. In this case, $C = |\mathcal{I}_i|$ and $D_I = 0$, and so $R_i^T/T \leq \Delta_i |\mathcal{I}_i| \sqrt{|A_i|}/\sqrt{T}$, recovering Theorem 4 of [124].

We now use Lemma 10 to prove Theorems 6 and 7:

Proof. We will show that for all $I \in \mathcal{I}_i$, $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$, and $a \in A(I)$,

$$\frac{1}{T}\left|R_i^{T,+}(\breve{I}, a) - k_{\breve{I},\breve{I}'}\ell_{\breve{I},\breve{I}'}R_i^{T,+}(\breve{I}', a)\right| \leq \delta_{\breve{I},\breve{I}'}\ell_{\breve{I},\breve{I}'}, \tag{A.3}$$

which, by Lemma 10, proves the theorem.

Fix $I \in \mathcal{I}_i$, $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$, and $a \in A(I)$. Firstly, for all $z \in Z_{\breve{I}}$ and $\sigma \in \Sigma$, by conditions (ii) and (iii) of Definition 2, we have

$$\begin{aligned}
\pi_{-i}^{\sigma}(z) &= \pi_c(z) \prod_{(I,a)\in X_{-i}(z)} \sigma(I,a) \\
&= \ell_{\breve{I},\breve{I}'}\pi_c(\phi(z)) \prod_{(I,a)\in X_{-i}(\phi(z))} \sigma(I,a) \\
&= \ell_{\breve{I},\breve{I}'}\pi_{-i}^{\sigma}(\phi(z))
\end{aligned} \tag{A.4}$$

and by condition (iv) of Definition 2, we similarly have

$$\pi_i^{\sigma}(z[\breve{I}], z) = \pi_i^{\sigma}(\phi(z)[\breve{I}'], \phi(z)) \tag{A.5}$$

and

$$\pi_i^{\sigma}(z[\breve{I}]a, z) = \pi_i^{\sigma}(\phi(z)[\breve{I}']a, \phi(z)). \tag{A.6}$$

We can then bound the positive part of the immediate counterfactual regret $R_i^{T,+}(\breve{I}, a)$:

$$\begin{aligned}
R_i^{T,+}(\breve{I}, a) &= \left(\sum_{t=1}^{T} r_i^t(\breve{I}, a)\right)^+ \\
&= \left(\sum_{t=1}^{T}\sum_{z\in Z_{\breve{I}}} \pi_{-i}^{\sigma}(z)(\pi_i^{\sigma}(z[\breve{I}]a, z) - \pi_i^{\sigma}(z[\breve{I}], z))u_i(z)\right)^+ \\
&\leq \Bigg(\sum_{t=1}^{T}\sum_{z\in Z_{\breve{I}}} \ell_{\breve{I},\breve{I}'}\pi_{-i}^{\sigma}(\phi(z))(\pi_i^{\sigma}(\phi(z)[\breve{I}']a, \phi(z)) \\
&\qquad\qquad - \pi_i^{\sigma}(\phi(z)[\breve{I}'], \phi(z)))(k_{\breve{I},\breve{I}'}u_i(\phi(z)) + \delta_{\breve{I},\breve{I}'})\Bigg)^+
\end{aligned}$$

by equations (A.4), (A.5), (A.6), and condition (i) of Definition 2

(A.7)

$$= \Bigg(\sum_{t=1}^{T} \sum_{z \in Z_{\breve{I}'}} \ell_{\breve{I},\breve{I}'} \pi^{\sigma}_{-i}(z) (\pi^{\sigma}_{i}(z[\breve{I}']a, z) - \pi^{\sigma}_{i}(z[\breve{I}'], z))(k_{\breve{I},\breve{I}'} u_i(z) + \delta_{\breve{I},\breve{I}'}) \Bigg)^{+}$$

since ϕ is a bijection

$$\leq \left(\sum_{t=1}^{T} \sum_{z \in Z_{\breve{I}'}} k_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'} \pi^{\sigma}_{-i}(z) (\pi^{\sigma}_{i}(z[\breve{I}]a, z) - \pi^{\sigma}_{i}(z[\breve{I}], z)) u_i(z) \right)^{+}$$
$$+ \left(\sum_{t=1}^{T} \sum_{z \in Z_{\breve{I}'}} \delta_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'} \pi^{\sigma}_{-i}(z) (\pi^{\sigma}_{i}(z[\breve{I}]a, z) - \pi^{\sigma}_{i}(z[\breve{I}], z)) \right)^{+}$$
$$\leq k_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'} R_i^{T,+}(\breve{I}', a) + \sum_{t=1}^{T} \delta_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'} \pi^{\sigma}_{-i}(\breve{I}')$$
$$\leq k_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'} R_i^{T,+}(\breve{I}', a) + T \delta_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'}, \tag{A.8}$$

where the last line follows because $\pi^{\sigma}_{-i}(\breve{I}') = \sum_{z \in Z_{\breve{I}'}} \pi^{\sigma}_{-i}(z[\breve{I}']) \leq 1$ in a perfect recall game $\breve{\Gamma}$. Similarly,

$$R_i^{T,+}(\breve{I}, a) \geq k_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'} R_i^{T,+}(\breve{I}', a) - T \delta_{\breve{I},\breve{I}'} \ell_{\breve{I},\breve{I}'}, \tag{A.9}$$

which together with equation (A.8) and dividing by T establishes (A.3), completing the proof. □

Note that Theorem 6 immediately follows from Theorem 7 since a well-formed game is skew well-formed with $\delta_{\breve{I},\breve{I}'} = 0$ for all $\breve{I}, \breve{I}' \in \breve{\mathcal{P}}(I)$.

A.3 Proofs of Supporting Lemmas for Theorem 8

Lemma 7. *For all states $s \in \mathcal{S}$, for all actions $a \in \mathcal{A}$, for all $\lambda \in (0, 2v_{\max}] \subset \mathbb{R}$, for all $c \in \mathbb{N}$, given a set $\mathcal{C}(s)$ of $c \in \mathbb{N}$ states generated according to $\mathcal{P}(\cdot \mid s, a)$, we have*

$$\mathbb{P}\left(\left| \left[\frac{1}{c} \sum_{s_i \in \mathcal{C}(s)} V_{d-1}(s_i) \right] - V_d(s, a) \right| \geq \lambda \right) \leq 2 \exp\left\{ -\lambda^2 c \,/\, 2v_{\max}^2 \right\}.$$

Proof. First note that $v_{\min} \leq V_d(s) \leq v_{\max}$, and since each game is zero-sum, $v_{\min} = -v_{\max}$. Also, clearly $\mathbb{E}_{s' \sim \mathcal{P}(\cdot \mid s,a)}[V_{d-1}(s')] = V_d(s, a)$ by definition. This lets us use a

special case of Hoeffding's Inequality, implied by [47, Theorem 2], which states that for a independent and identically distributed random sample $X_1, \ldots, X_c$ it holds that

$$\mathbb{P}\left(\left|\frac{1}{c}\sum_{i=1}^{c} X_i - \mathbb{E}[X]\right| \geq \lambda\right) \leq 2\exp\left\{-2\lambda^2 c^2 \;/\; \sum_{i=1}^{c}(b-a)^2\right\}, \tag{A.10}$$

provided $a \leq X_i \leq b$. Applying this bound, setting $b-a$ to $2\,v_{\max}$ and simplifying finishes the proof. □

Proposition 1. *For all $d \in \mathbb{N}$, for a state $s \in \mathcal{S}$, if $\left|\hat{V}_d(s,a) - V_d(s,a)\right| < \lambda$ holds for all $a \in \mathcal{A}$, then $\left|\hat{V}_d(s) - V_d(s)\right| < \lambda$.*

Proof. Recall, $V_d(s) = \max_{a\in\mathcal{A}} V_d(s,a)$ and $\hat{V}_d(s) = \max_{a\in\mathcal{A}} \hat{V}_d(s,a)$ for any state $s \in \mathcal{S}$. Also define $a^* = \arg\max_{a\in\mathcal{A}} V_d(s,a)$ and $\hat{a}^* = \arg\max_{a\in\mathcal{A}} \hat{V}_d(s,a)$. Now, it holds that

$$\hat{V}_d(s) - V_d(s) = \hat{V}_d(s,\hat{a}^*) - V_d(s) \leq [V_d(s,\hat{a}^*) + \lambda] - V_d(s) \leq [V_d(s,a^*) + \lambda] - V_d(s) = \lambda,$$

and also

$$\hat{V}_d(s) - V_d(s) = \hat{V}_d(s,\hat{a}^*) - V_d(s) \geq \hat{V}_d(s,a^*) - V_d(s) \geq [V_d(s,a^*) - \lambda] - V_d(s) = -\lambda,$$

hence $\left|\hat{V}_d(s) - V_d(s)\right| < \lambda$. □

Appendix B

Best Response Algorithms

Here, we describe the best response algorithms use to compute the **exploitability** value ϵ_σ reported in in the CFR and MCCFR experiments.

We present here two versions of the best response algorithm. The first form has been very well described in previous work [53] (for an introduction, see [102, Chapter 3]), but will help in understanding the generalized form, so we summarize it here. The generalized form is required for games that have hidden or partially hidden actions, such as Goofspiel, Princess and Monster, Latent Tic-Tac-Toe and Phantom Tic-Tac-Toe.

Given a profile $\sigma = (\sigma_1, \sigma_2)$, *e.g.,* CFR's average strategy after T iterations, the best response algorithm computes $\max_{\sigma'_1 \in \Sigma_1} u(\sigma'_1, \sigma_2)$ or $\max_{\sigma'_2 \in \Sigma_2} u(\sigma_1, \sigma'_2)$, or both when computing ϵ_σ. To do this, expectimax is used where one player $(-i)$ is considered a fixed player and at each node $h \in H_{-i}$ plays $\sigma_{-i}(I)$ where $h \in I$; the other player (i), at information sets where $P(I) = i$, computes a strategy $\sigma^*_i = \max_{a \in A(I)} u(\sigma^*_{i, I \to a}, \sigma_{-i})$. Since a pure strategy best response always exists, the best response algorithm is a slightly more complex (due to information sets) application of expectimax in a single-player game, and is summarized in Algorithm 8.

In many games, such as Bluff and Poker, the outcomes of the chance events only affect the payoffs at the leaves, not the actions that are available to the players. Further, the chance outcome can be separated into public outcomes observable by all players, private outcomes observable only to i or only to $-i$. In these games, the best response algorithm can simply traverse the public tree (including the public chance actions) and compute the opponent's action distribution D_{-i} at each of their information sets for each *full outcome* private to the opponent $o \in \mathcal{C}_{-i}$, where each $o \in \mathcal{C}_{-i}$ is a sequence of chance outcomes occurring at nodes private to $-i$. The private chance outcomes of the opponent are never truly assigned to a history until Line 5 where $o \circ h$ represents turning the public action

Algorithm 8 Best Response Algorithm for Bluff and Poker Games

Initialize chance outcome vector $\vec{\pi}_{-i} = (\pi_c(o))_{o \in \mathcal{C}_{-i}}$
function BestResponse(h, i, $\vec{\pi}_{-i}$):
if h is terminal **then**
    $D_{-i} \leftarrow$ Normalize($\vec{\pi}_{-i}$)
    $u \leftarrow \sum_{o \in \mathcal{C}_{-i}} u(o \circ h) \cdot D_{-i}(o)$
    return u
else if h is a public chance node or private chance node belonging to i **then**
    return $\sum_{a \in A(I)}$ BestResponse$(ha, i, \vec{\pi}_{-i})$
else if h is a private chance node belonging to $-i$ **then**
    Choose any $a \in A(I)$ as a dummy outcome and **return** BestResponse$(ha, i, \vec{\pi}_{-i})$
end if
Let D_{-i} be the opponent's action distribution
$\vec{\pi}'_{-i} \leftarrow \vec{\pi}_{-i}$
$v \leftarrow -\infty$
$v_\sigma[a] \leftarrow 0$ for all $a \in A(I)$
$w[a] \leftarrow 0$ for all $a \in A(I)$
for $a \in A(I)$ **do**
    if $P(h) = -i$ **then**
        $(w[a], \vec{\pi}'_{-i}) \leftarrow$ ComputeWeight$(I, a, \vec{\pi}_{-i})$
    end if
    $v_\sigma[a] \leftarrow$ BestResponse$(ha, i, \vec{\pi}'_{-i})$
    if $P(h) = i$ **and** $v_\sigma[a] > v$ **then**
        $v \leftarrow v_\sigma[a]$
    end if
end for
if $P(h) = -i$ **then**
    $D_{-i} \leftarrow$ Normalize(w)
    $v \leftarrow \sum_{a \in A(h)} D_{-i}(a) \cdot v_\sigma[a]$
end if
return v

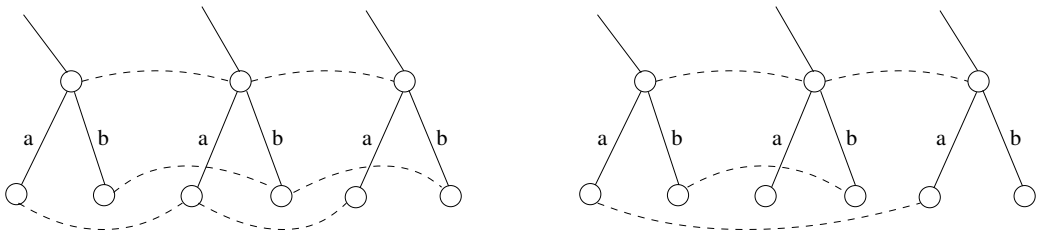


Figure B.1: When all player's actions are fully observable (left), such as in Bluff and Poker, taking an action a from one information set I will take you to the same I', regardless of the prefix history a was taken from. However, this may not be the case in games with only partially observable actions (right), such as Goofspiel.

sequence h into a full terminal history by assigning the private chance outcomes o. To determine the opponent's strategy distribution at an opponent node given only a sequence of public actions, the ComputeWeight function first computes the new opponent reach vector $\vec{\pi}_{-i}$ by multiplying each entry by $\sigma(I, a)$ for the information set I matching h and o, then assigns a weight $w[a] = \sum_{o \in \mathcal{C}_{-i}} \vec{\pi}'_{-i}[o]$. Normalizing these weights at Line 27 will give probability that they will play a given all their possible private outcomes by Bayes' rule.

Now, suppose the game has hidden or partially observable actions. In other words, when some player takes an action a from information set I, the next information set will depend on the particular prefix history $h \in I$, possibly because h has some hidden actions taken by the opponent. The difference is illustrated in Figure B.1. In our version of Goofspiel, the players are only informed whether they won or lost, the card they played is never revealed. Suppose player 1 plays a card face down. It's not player 2's turn and they respond by playing a 7; player 2 can either win the point card or lose it (two different information sets), but it will depend on what card player 1 played.

We now describe a generalized version of the best response algorithm that will work for games with hidden actions. As before, we have a player i computing the best response and a fixed player $-i$. At each information set, two additional arrays are required to compute the expected value of and action $a \in A(I) : t_I[a]$ and $b_I[a]$. The algorithm requires three passes. The first pass set $t_I[a] = b_I[a] = 0$ for all I and collects the depths of information sets $I \in \mathcal{I}_i$. This list of depths is then sorted in decreasing order so that the highest depths are first. There are several second passes, described in Algorithm 9, which essentially proceeds by iterative-shallowing: each one repeatedly called with depth d taken from sorted list in decreasing order. The last pass returns the expected value at each node by computing a linear combination at chance and opponent nodes, and selecting the maximum

of $\frac{t[a]}{b[a]}$ at nodes belonging to i.

Algorithm 9 Generalized Expectimax Best Response (GEBR)

```
1:  function GEBR-Pass2(h, i, d, l, π_{-i}):
2:  if h is terminal then
3:      return u_i(z)
4:  else if h is a chance node then
5:      return Σ_{a∈A(h)} σ_c(a|h) · GEBR-Pass2(ha, i, d, l + 1, π_{-i} · σ_c(a|h))
6:  end if
7:  Let I be the information set containing h
8:  v ← 0
9:  if P(I) = i and l > d then
10:     Choose a = arg max_{a∈A(I)} t[a]/b[a]
11:     return GEBR-Pass2(ha, i, d, l + 1, π_{-i})
12: end if
13: for a ∈ A(I) do
14:     π'_{-i} ← π_{-i}
15:     if P(I) = −i then
16:         π'_{-i} ← π_{-i} · σ(I, a)
17:     end if
18:     v' ← GEBR-Pass2(ha, i, d, l + 1, π'_{-i})
19:     if P(I) = −i then
20:         v ← v + σ(I, a) · v'
21:     else if P(I) = i and l = d then
22:         t_I[a] ← t_I[a] + v' · π_{-i}
23:         b_I[a] ← b_I[a] + π_{-i}
24:     end if
25: end for
26: return v
```

In the second passes, the expected values below each (I, a) are computed by traversing over each prefix $h \in I$ for all I at depth d. Lines 22 and 23 compute the expected value below (I, a) by accumulating a sum of the opponent's reach probability given each prefix $h \in I$. When the pass reaches a node whose depth is higher than d, then the best action has been computed by a previous update pass and the best one is chosen on Line 10.

While this best response algorithm is much slower in practice, it computes the correct value for any finite two-player zero-sum extensive-form game.