Efficient Nash Equilibrium Approximation through Monte Carlo Counterfactual Regret Minimization

Michael Johanson, Nolan Bard, Marc Lanctot, Richard Gibson, and Michael Bowling
University of Alberta
Edmonton, Alberta
{johanson,nolan,lanctot,rggibson,bowling}@cs.ualberta.ca

ABSTRACT

Recently, there has been considerable progress towards algorithms for approximating Nash equilibrium strategies in extensive games. One such algorithm, Counterfactual Regret Minimization (CFR), has proven to be effective in two-player zero-sum poker domains. While the basic algorithm is iterative and performs a full game traversal on each iteration, sampling based approaches are possible. For instance, chance-sampled CFR considers just a single chance outcome per traversal, resulting in faster but less precise iterations. While more iterations are required, chance-sampled CFR requires less time overall to converge. In this work, we present new sampling techniques that consider sets of chance outcomes during each traversal to produce slower, more accurate iterations. By sampling only the public chance outcomes seen by all players, we take advantage of the imperfect information structure of the game to (i) avoid recomputation of strategy probabilities, and (ii) achieve an algorithmic speed improvement, performing $O(n^2)$ work at terminal nodes in $O(n)$ time. We demonstrate that this new CFR update converges more quickly than chance-sampled CFR in the large domains of poker and Bluff.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*

General Terms

Algorithms

Keywords

Economic paradigms::Game theory (cooperative and non-cooperative)

1. INTRODUCTION

Extensive games are an intuitive formalism for modelling interactions between agents in a sequential decision making setting. One solution concept in such domains is a Nash equilibrium. In two-player zero-sum domains, this is equivalent to a minmax strategy, which minimizes each agent's expected worst-case performance. For games of moderate size, such a strategy can be found using linear programming [5]. For larger games, techniques such as Counterfactual Regret Minimization (CFR) [10] and the Excessive Gap Technique [3] require less memory than linear programming and are capable of finding an equilibrium in games (also known as **solving** a game) with up to 10^{12} game states.

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

CFR is an iterative procedure that resembles self-play. On each iteration, CFR performs a full game tree traversal and updates its entire strategy profile to minimize regret at each decision. Theoretical bounds suggest that the procedure takes a number of iterations at most quadratic in the size of a player's strategy [10, Theorem 4]. Thus, as we consider larger games, not only are more iterations required to converge, but each traversal becomes more time consuming. A variant known as **Chance-Sampled (CS)** CFR [6, 10] samples one set of chance outcomes per iteration and traverses only the corresponding portion of the game tree. Compared to the basic algorithm, this sampling procedure results in faster but less precise strategy updates. In large games, the drastic reduction in per-iteration time cost outweighs the increased number of iterations required for convergence to an optimal strategy.

While CS considers only a single set of chance outcomes per iteration, recent work [4] towards fast best-response computation has shown that tree traversal and evaluation can be accelerated by simultaneously considering sets of information sets for each player. This allows for the caching and reuse of computed values, and also allows a fast terminal node evaluation in which $O(n^2)$ work can often be done in $O(n)$ time. While best response calculation in large games was previously considered intractable, the new technique was shown to perform the computation in just over one day [4].

In this paper, we apply this new tree traversal to CFR, resulting in three new sampling variants: **Self-Public Chance Sampling (SPCS)**, **Opponent-Public Chance Sampling (OPCS)**, and **Public Chance Sampling (PCS)**. The new techniques reverse the previous trend in that they advocate less sampling: a small number of slow iterations, each updating a large number of information sets, yielding precise strategy updates while reusing computed values. In particular, PCS takes advantage of the computation reuse and fast terminal node evaluation used in accelerating the best response computation. We will prove the convergence of the new techniques, investigate their qualities, and demonstrate empirically that PCS converges more quickly to an equilibrium than CS in both poker and the game of Bluff.

2. BACKGROUND

An extensive game is a general model of sequential decision-making with imperfect information. Extensive games consist primarily of a game tree whose nodes correspond to **histories** (sequences) of actions $h \in H$. Each non-terminal history, h, has an associated **player** $P(h) \in N \cup \{c\}$ (where N is the set of players and c denotes **chance**) that selects an **action** $a \in A(h)$ at that history h. When $P(h) = c$, $f_c(a|h)$ is the (fixed) probability of chance generating action a at h. We call h a **prefix** of history h', written $h \sqsubseteq h'$, if h' begins with the sequence h. Each **terminal history** $z \in Z \subset H$ has associated **utilities** for each player i, $u_i(z)$. In **imperfect information** games, histories are partitioned into **information sets** $I \in \mathcal{I}_i$ representing different game states that player i cannot distinguish between. For example, in poker, player i does not see the opponents' private cards, and thus all histories differing only in the private cards dealt to the opponents are in the same information set for player i. For histories $h, h' \in I$, the actions available at h and h' must be the same, and we denote this action set by $A(I)$. We also assume **perfect recall** that guarantees players always remember information that was revealed to them and the order in which it was revealed.

A **strategy for player *i***, σ_i, is a function that maps each $I \in \mathcal{I}_i$ to a probability distribution over $A(I)$. We denote Σ_i as the set of all strategies for player i. A **strategy profile** is a vector of strategies $\sigma = (\sigma_1, \ldots, \sigma_{|N|})$, one for each player. We let σ_{-i} refer to the strategies in σ excluding σ_i.

Let $\pi^\sigma(h)$ be the probability of history h occurring if all players choose actions according to σ. We can decompose

$$\pi^\sigma(h) = \prod_{i \in N} \pi_i^\sigma(h) \prod_{\substack{h'a \sqsubseteq h \\ P(h')=c}} f_c(a|h')$$

into each player's and chance's contribution to this probability. Here, $\pi_i^\sigma(h)$ is the contribution from player i when playing according to σ_i. Let $\pi_{-i}^\sigma(h)$ be the product of all players' contribution (including chance) except that of player i. Furthermore, let $\pi^\sigma(h, h')$ be the probability of history h' occurring, given h has occurred with $\pi_i^\sigma(h, h')$, and $\pi_{-i}^\sigma(h, h')$ defined similarly.

Given a strategy profile, σ, we define a player's **best response** as a strategy that maximizes their expected payoff, assuming all other players play according to σ. The **best-response value** for player i is the value of that strategy, $b_i(\sigma_{-i}) = \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i})$. A strategy profile σ is an **ϵ-Nash equilibrium** if no player can deviate from σ and gain more than ϵ; *i.e.* $u_i(\sigma) + \epsilon \geq \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i})$ for all $i \in N$. If $\epsilon = 0$, then σ is a **Nash equilibrium** and every player is playing a best response.

In this paper, we will focus on **two-player zero-sum** games: $N = \{1, 2\}$ and $u_1(z) = -u_2(z)$ for all $z \in Z$. In this case, the **exploitability** of σ, $\epsilon_\sigma = (b_1(\sigma_2) + b_2(\sigma_1))/2$, measures how much σ loses to a worst case opponent when players alternate positions. A Nash equilibrium has an exploitability of 0.

Lastly, define $\mathcal{C} = \{h \in H : P(h) = c\}$ to be the set of all histories where it is chance's turn to act. We will assume that $\mathcal{C}$ can be partitioned into three sets with respect to player i: $\mathcal{S}_i$, $\mathcal{O}_i$, and $\mathcal{P}$. Each set contains the histories h whose actions $a \in A(h)$, or **chance events**, are observable only by player i ($\mathcal{S}_i$), only by player i's opponent ($\mathcal{O}_i$), or by both players ($\mathcal{P}$). We refer to chance events occurring at $h \in \mathcal{S}_i \cup \mathcal{O}_i$ as **private** and to chance events occurring at $h \in \mathcal{P}$ as **public**. In addition, we assume that the actions available to the players throughout the game are independent of the private chance events. These two assumptions hold for a large class of games, including poker as well as any Bayesian game with observable actions [8] (*e.g.*, Bluff or negotiation games); furthermore, games can often be modified by adding additional chance actions to satisfy the property.

2.1 Counterfactual Regret Minimization

Counterfactual Regret Minimization (CFR) resembles a self-play algorithm where we iteratively obtain strategy profiles σ^t based on regret values accumulated throughout previous trials. At each information set $I \in \mathcal{I}_i$, the expected value for player i at I under the current strategy is computed, assuming player i plays to reach I. This expectation is the **counterfactual value** for player i,

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

where Z_I is the set of terminal histories passing through I and $z[I]$ is the prefix of z contained in I. For each action $a \in A(I)$, these values determine the **counterfactual regrets** at iteration t, $r_i^t(I, a) = v_i(\sigma_{(I \to a)}^t, I) - v_i(\sigma^t, I)$, where $\sigma_{(I \to a)}$ is the profile σ except at I, action a is always taken. The regret $r_i^t(I, a)$ measures how much player i would rather play action a at I than play σ^t. The counterfactual regrets are accumulated and σ^t is updated by applying regret matching [2, 10] to the accumulated regrets. Regret matching is a regret minimizer; *i.e.*, over time, the average of the counterfactual regrets approaches 0. Minimizing counterfactual regret at each information set minimizes the **average overall regret** [10, Theorem 3], defined by

$$R_i^T = \max_{\sigma' \in \Sigma_i} \frac{1}{T} \sum_{t=1}^{T} \left(u_i(\sigma', \sigma_{-i}^t) - u_i(\sigma_i^t, \sigma_{-i}^t) \right).$$

It is well-known that in a two-player zero-sum game, minimizing average overall regret implies that the average profile $\overline{\sigma}^T$ is an approximate equilibrium. CFR produces an ϵ-Nash equilibrium in $O(|H||\mathcal{I}_i|/\epsilon^2)$ time [10, Theorem 4].

Rather than computing the exact counterfactual values on every iteration, one can instead sample the values using **Monte Carlo CFR (MCCFR)** [6]. **Chance-sampled (CS) CFR** [10] is an instance of MCCFR that considers just a single set of chance outcomes per iteration. In general, let $\mathcal{Q}$ be a set of subsets, or **blocks**, of the terminal histories Z such that the union of all blocks spans Z. For CS, $\mathcal{Q}$ is the partition of Z where two histories belong to the same block if and only if no two chance events differ. In addition, a probability distribution over $\mathcal{Q}$ is required and a block $Q \in \mathcal{Q}$ is sampled on each iteration, giving us the **sampled counterfactual value** for player i,

$$\tilde{v}_i(\sigma, I) = \sum_{z \in Z_I \cap Q} u_i(z) \pi_{-i}^\sigma(z[I]) \pi^\sigma(z[I], z)/q(z),$$

where $q(z)$ is the probability that z was sampled. In CS, we sample the blocks according to the likelihood of the chance events occurring, so that

$$q(z) = \prod_{\substack{ha \sqsubseteq z \\ h \in \mathcal{C}}} f_c(a|h).$$

The counterfactual regrets are then measured according to these sampled values, as opposed to "vanilla CFR" that uses the true values $v_i(\sigma, I)$. Sampling reduces enumeration to the smaller subset Q rather than all of Z, decreasing the amount of time required per iteration. For a fixed ϵ, CS requires more iterations than vanilla CFR to obtain an ϵ-Nash equilibrium; however, the overall computing time for CS is lower in poker games [9, Appendix A.5.2].

2.2 Accelerated Traversal and Evaluation

A recent paper describes how to accelerate the computation of the best response value in large extensive form games [4]. This technique traverses a game's **public tree**, which represents the state of the game visible to all players. The authors observe that each player's strategy must be independent of the other player's private information. As such, a player's action probabilities can be computed just once while considering the opponent's entire set of possible private states in one traversal.

In addition, the authors describe an efficient terminal node evaluation that considers a range of n information sets for each player in tandem. If the game's payoffs exhibit structure, then it may be possible to exploit this structure and reduce a naive $O(n^2)$ computation to $O(n)$. Examples of structured payoffs include games where utilities are affected by only certain factors within the players' information sets, such as in a negotiation game, and games where information sets can be ranked from weakest to strongest, such as in poker. This algorithmic speedup is not being used in any of the previously published equilibrium solvers. In Section 3, we describe how to use these ideas to produce a new equilibrium solver that outperforms the current state of the art.

2.3 Domains: Poker and Bluff

The Game of Poker. Our main poker game of interest is heads-up (*i.e.*, two-player) limit Texas hold'em poker, or simply **Texas hold'em**. The game uses a standard 52 card deck and consists of 4 betting rounds. In the first round, the **pre-flop**, each player is dealt two private cards. For subsequent rounds – in order, the **flop**, **turn**, and **river** – public community cards are revealed (3 at the flop and 1 at each of the turn and river). During each round, players sequentially take one of three actions: **fold** (forfeit the game), **call** (match the previous bet), or **raise** (increase the bet). There is a maximum of 4 raises per round, each with a fixed size, where the size is doubled on the final two rounds. If neither player folds, then the player with the highest ranked poker hand wins all of the bets.

Texas hold'em contains approximately 3.2×10^{14} information sets. The large size of the game makes an equilibrium computation intractable for all known algorithms; CFR would require more than ten petabytes of RAM and hundreds of CPU-years of computation. A common approach is to use state-space abstraction to produce a similar game of a tractable size by merging information sets or restricting the action space [1]. In Section 4, we consider several abstractions of Texas hold'em and two new variants of Texas hold'em that are small enough to compute equilibrium solutions using CFR without abstraction. The first new variant is **[2-1] hold'em**. The game is identical to Texas hold'em, except consists of only the first two betting rounds, the pre-flop and flop, and only one raise is allowed per round. This reduces the size of the game to 16 million information sets. Similarly, **[2-4] hold'em** has just two rounds, but the full four raises are allowed per round, resulting in 94 million information sets in total. In both [2-1] hold'em and [2-4] hold'em, the size of a raise doubles from the pre-flop to the flop.

The Game of 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 1 to 5 and a star: $\star$. Each player i rolls D_i of these dice and looks at them without showing them to their opponent. On each round, players alternate by bidding on the outcome of all dice in play until one player claims that the other is bluffing (*i.e.*, claims that the bid does not hold). A bid consists of a **quantity** of dice and a **face** value. A face of $\star$ is considered "wild" and counts as matching any other face. For example, the bid 2-5 represents the claim that there are at least two dice with a face of 5 or $\star$ among both players' dice. To place a new bid, the player must increase either the quantity or face value of the current bid; in addition, lowering the face is allowed if the quantity is increased. The player calling bluff wins the round if the opponent's last bid is incorrect, and loses otherwise. The losing player removes one of their dice from the game and a new round begins, starting with the player who won the previous round. 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.

In this paper, we restrict ourselves to the case where $D_1 = D_2 = 2$, a game containing 352 million information sets. Note that since Bluff(2,2) is a multi-round game, the expected values of Bluff(1,1) are precomputed for payoffs at the leaves of Bluff(2,1), which is then solved for leaf payoffs in the full Bluff(2,2) game.

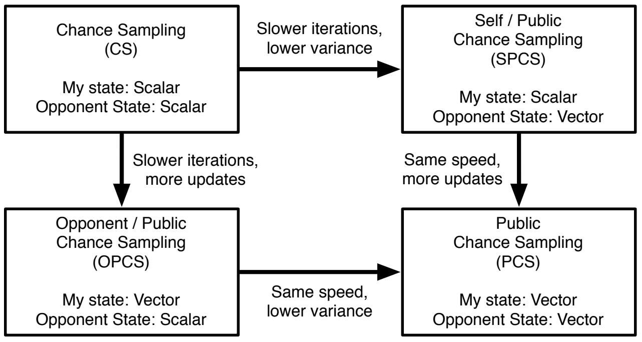


Figure 1: Relationship between MCCFR variants

3. NEW MONTE CARLO CFR VARIANTS

Before presenting our new CFR update rules, we will begin by providing a more practical description of chance-sampled CFR. On each iteration, we start by sampling all of chance's actions: the public chance events visible to each player, as well as the private chance events that are visible to only a subset of the players. In poker, this corresponds to randomly choosing the public cards revealed to the players, and the private cards that each player is dealt. In the game of Bluff, there are no public chance events, and only private chance events are sampled for each player. Next, we recursively traverse the portion of the game tree that is reachable given the sampled chance events, and explore all of the players' actions. On the way from the root to the leaves, we pass forward two scalar values: the probability that each player would take actions to reach their respective information sets, given their current strategy and their private information. On the way back from the leaves to the root, we return a single scalar value: the sampled counterfactual value $\tilde{v}_i(\sigma, I)$ for player i. At each choice node for player i, these values are all that is needed to calculate the regret for each action and update the strategy. Note that at a terminal node $z \in Z$, it takes $O(1)$ work to determine the utility for player i, $u_i(z)$.

We will now describe three different methods of sampling chance events that have slower iterations, but do more work on each iteration. Figure 1 shows the relationship between CS and these three new variants, all of which belong to the MCCFR family [6] of update rules.

Opponent-Public Chance Sampling. Consider a variation on CS, where instead of sampling at every chance node, we sample an action for just the opponent's chance and the public chance events while enumerating all of the possible outcomes at our private chance events. We will call this variant Opponent-Public Chance Sampling (OPCS). This can be formalized within the MCCFR framework by letting $\mathcal{Q}$ be the partition of Z such that two histories fall into the same block if and only if the actions taken at opponent and public chance events match. The probability that z is sampled is then

$$q(z) = \prod_{\substack{ha \sqsubseteq z \\ h \in \mathcal{O}_{-i} \cup \mathcal{P}}} f_c(a|h).$$

Naively, we could use the same recursive tree walk that we used for CS to perform this update, by doing one tree walk for each of our private chance outcomes in turn. However, this update allows us to traverse the sampled portion of the game tree in a much more efficient way. Since our opponent does not observe our private chance events, their strategy and choice of actions, given their single sampled chance event, cannot depend on which information set we are in. This means that we can update all of our information sets that are consistent with the current game state and the sampled public chance events at the same time, thus amortizing the cost of walking the tree over many updates. This can be achieved by a new recursive tree walk that passes forwards a vector for us (our probability of reaching the current game state with each of our private chance outcomes) and a scalar for the opponent (their probability of reaching the current game state with their single sampled private chance outcome), and returns a vector of values (our counterfactual value for each of our private chance outcomes).

At terminal nodes, we must evaluate n possible game states, each consisting of a different private chance outcome for us and one chance outcome for the opponent. This requires $O(n)$ time. In comparison to CS, each iteration of OPCS is slower, but performs more work by updating a much larger number of information sets.

Self-Public Chance Sampling. In OPCS, we enumerate over all of our possible private chance outcomes. Alternatively, we can instead enumerate over all of our opponent's private chance outcomes while sampling our own private chance outcomes and the public chance outcomes. We will call this variant Self-Public Chance Sampling (SPCS). This can similarly be formalized by defining $\mathcal{Q}$ to be the partition of Z that separates histories into different blocks whenever the actions taken at our private or public chance events differ, where

$$q(z) = \prod_{\substack{ha \sqsubseteq z \\ h \in \mathcal{S}_i \cup \mathcal{P}}} f_c(a|h)$$

is the probability of sampling terminal history z.

As in OPCS, we can use an efficient recursive tree walk to perform this update. Since we cannot observe the opponent's private chance events, our strategy and choice of actions cannot depend on which information set they are in. Thus, when computing our counterfactual value, we will consider every possible private chance outcome for our opponent. Doing so forms a more accurate estimate of the true counterfactual value for our sampled outcome, compared to the noisy estimate CS and OPCS obtain through one sampled opponent private chance outcome. The SPCS tree walk passes forward a scalar for ourselves (the probability of reaching the current game state with our single chance outcome) and a vector for the opponent (their probabilities of reaching the current game state with each of their private chance outcomes), and returns a scalar (the counterfactual value for our sampled outcome).

At terminal nodes, we must evaluate up to n possible game states, formed by our single chance outcome and up to n possible chance outcomes for the opponent. This requires $O(n)$ time. In comparison to CS, each iteration is slower and performs the same number of updates to the strategy, but each update is based off of much more precise estimates.

Public Chance Sampling. We will now introduce the core contribution of this work, called Public Chance Sampling (PCS), that combines the advantages of both of the previous two updates, while taking advantage of efficient terminal node evaluation to keep the time cost per iteration in $O(n)$. In PCS, we sample only the public chance events, and consider all possible private chance events for ourself and for the opponent. In other words, we define $\mathcal{Q}$ to be the partition of Z that separates histories into different blocks whenever the actions taken at a public chance event differ, where

$$q(z) = \prod_{\substack{ha \sqsubseteq z \\ h \in \mathcal{P}}} f_c(a|h)$$

is the probability of sampling $z \in Z$.

PCS relies on the property that neither us nor our opponent can observe the other's private chance events, and so the action probabilities for each remain the same across the other's private information. Thus, we can perform a CFR update through a recursive tree walk with the following structure. On the way from the root to the leaves, we will pass forwards two vectors: one containing the probabilities of us and one containing the probabilities of the opponent reaching the current game state, for each player's n possible private chance outcomes. On the way back, we will return a vector containing the counterfactual value for each of our n information sets.

At the terminal nodes, we seemingly have an $O(n^2)$ computation, as for each of our n information sets, we must consider all n of the opponent's possible private outcomes in order to compute our utility for that information set. However, if the payoffs at terminal nodes are structured in some way, we can often reduce this to an $O(n)$ evaluation that returns exactly the same value as the $O(n^2)$ evaluation [4]. Doing so gives PCS the advantage of both SPCS (accurate strategy updates) and OPCS (many strategy updates) for the same evaluation cost of either.

3.1 Algorithm

The three new chance-sampling variants, along with CS, are shown in Algorithm 1. The WalkTree function traverses down the game tree by recursively concatenating actions, starting with the empty history $h = \emptyset$, and updates player i's regrets and average strategy on the way back up. Two vectors are maintained, one for player i, $\vec{\pi}_i$, and one for the opponent, $\vec{\pi}_{-i}$. These vectors keep track of the probabilities of reaching each information set consistent with the current history h, with each element corresponding to a different private chance outcome for that player. In CS, both vectors have length one (*i.e.*, are scalars). In OPCS, $\vec{\pi}_{-i}$ has length one because the opponent's private chance events are being sampled. Similarly, in SPCS, $\vec{\pi}_i$ has length one.

When the current sequence h is a terminal history (line 6), the utility is computed and returned. At line 7, $\vec{f}_{c,i}(h)$ and $\vec{f}_{c,-i}(h)$ are the vectors corresponding to the probability distribution over player i's and the opponent's private chance outcomes, respectively, and $\odot$ represents element-wise vector multiplication. Again, one or both vectors may have length one depending on the selected variant, in which case the single element is always 1. For OPCS and PCS, $\vec{u}_i$ is a vector containing a utility for each of player i's private outcomes; for SPCS and CS, $\vec{u}_i$ is a length one vector corresponding to the utility for player i's sampled private outcome. PCS uses the $O(n^2)$ to $O(n)$ algorithmic improvement to compute $\vec{u}_i$, which will be described in Section 3.2.

Chance events are handled by lines 12 to 18. When one of the four conditions at line 12 holds, we are at a chance event that is to be sampled; otherwise, we consider all possible chance events at h. In the latter case, we must take a dummy action (line 16) simply to continue traversing the tree. This action has no effect on the remainder of the tree walk due to our assumption that player actions are independent of private chance events.

Lines 19 to 42 handle the cases where h is a decision node for one of the players. First, lookupInfosets(h) retrieves all of the information sets consistent with h and the current player $P(h)$'s range of possible private outcomes, whether sampled ($|\vec{I}| = 1$) or not. Next, at line 21, regret matching [2, 10] determines the current strategy $\vec{\sigma}$, a vector of action probabilities for each retrieved information set (and thus, in general, a vector of vectors). Regret matching assigns action probabilities according to

$$\sigma[a][I] = \begin{cases} r_I^+[a] / \sum_{b \in A(I)} r_I^+[b] & \text{if } \sum_{b \in A(I)} r_I^+[b] > 0 \\ 1/|A(I)| & \text{otherwise,} \end{cases}$$

Algorithm 1 PCS Algorithm

```
1: Require: a variant $v \in \{\text{CS}, \text{OPCS}, \text{SPCS}, \text{PCS}\}$.
2: Initialize regret tables: $\forall I, r_I[a] \leftarrow 0$.
3: Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$.
4:
5: function WalkTree($h$, $i$, $\vec{\pi}_i$, $\vec{\pi}_{-i}$):
6:     if $h \in Z$
7:         return $\vec{f}_{c,i}(h) \odot \vec{u}_i \left(h \mid \vec{\pi}_{-i} \odot \vec{f}_{c,-i}(h)\right)$
8:     end if
9:     if     ($v$ = PCS and $h \in \mathcal{P}$)
10:        or ($v$ = SPCS and $h \in \mathcal{S}_i \cup \mathcal{P}$)
11:        or ($v$ = OPCS and $h \in \mathcal{O}_i \cup \mathcal{P}$)
12:        or ($v$ = CS and $h \in \mathcal{C}$)
13:        Sample outcome $a \in A(h)$ with probability $f_c(a|h)$
14:        return WalkTree($ha$, $i$, $\vec{\pi}_i$, $\vec{\pi}_{-i}$)
15:     else if $h \in \mathcal{C}$
16:        Select dummy outcome $a \in A(h)$
17:        return WalkTree($ha$, $i$, $\vec{\pi}_i$, $\vec{\pi}_{-i}$)
18:     end if
19:     $\vec{I} \leftarrow$ lookupInfosets($h$)
20:     $\vec{u} \leftarrow \vec{0}$
21:     $\vec{\sigma} \leftarrow$ regretMatching($\vec{I}$)
22:     for each action $a \in A(h)$ do
23:        if $P(h) = i$
24:           $\vec{\pi}'_i \leftarrow \vec{\sigma}[a] \odot \vec{\pi}_i$
25:           $\vec{u}' \leftarrow$ WalkTree($ha$, $i$, $\vec{\pi}'_i$, $\vec{\pi}_{-i}$)
26:           $\vec{m}[a] \leftarrow \vec{u}'$
27:           $\vec{u} \leftarrow \vec{u} + \vec{\sigma}[a] \odot \vec{u}'$
28:        else
29:           $\vec{\pi}'_{-i} \leftarrow \vec{\sigma}[a] \odot \vec{\pi}_{-i}$
30:           $\vec{u}' \leftarrow$ WalkTree($ha$, $i$, $\vec{\pi}_i$, $\vec{\pi}'_{-i}$)
31:           $\vec{u} \leftarrow \vec{u} + \vec{u}'$
32:        end if
33:     end for
34:     if $P(h) = i$
35:        for $I \in \vec{I}$ do
36:           for $a \in A(I)$ do
37:              $r_I[a] \leftarrow r_I[a] + m[a][I] - u[I]$
38:              $s_I[a] \leftarrow s_I[a] + \pi_i[I]\sigma[a][I]$
39:           end for
40:        end for
41:     end if
42:     return $\vec{u}$
43:
44: function Solve():
45:     for $t \in \{1, 2, 3, \cdots\}$ do
46:        for $i \in N$ do
47:           WalkTree($\emptyset$, $i$, $\vec{1}$, $\vec{1}$)
48:        end for
49:     end for
```

where $r_I^+[a] = \max\{r_I[a], 0\}$. We then iterate over each action $a \in A(h)$, recursively obtaining the expected utilities for a at each information set (line 25 or 30). When $P(h) = i$, these utilities are stored (line 26) and used to update the regret at each information set (line 37), while the current strategy $\vec{\sigma}$ weights both the returned expected utility at h (line 27) and the average strategy update (line 38). Note that at line 31, we do not weight $\vec{u}'$ by $\vec{\sigma}[a]$ since the opponent's reaching probabilities are already factored into the utility computation (line 7).

After iterating over the outer loop of Solve() (line 45) for many iterations, an ϵ-Nash equilibrium is obtained from the accumulated strategies: $\bar{\sigma}(I, a) = s_I[a] / \sum_{b \in A(I)} s_I[b]$.

3.2 Efficient Terminal Node Evaluation

We now describe how PCS computes a vector of expected utilities $\vec{u}_i(h \mid \vec{\pi}_{-i})$ at line 7 for player i's n private outcomes in $O(n)$ time. As we have already noted, Johanson *et al.* [4] gave a detailed description for how to do this in poker. In this section, we will describe an efficient terminal node evaluation for Bluff(D_1, D_2).

Every game ends with one player calling bluff, and the payoffs ($+1$ or -1) are determined solely by whether or not the last bid holds. Let x-y be the last such bid. We now must discriminate between cases where there are less than and where there are at least x dice showing face y or $\star$.

At the terminal history h, we have a vector of reach probabilities $\vec{\pi}_{-i}$ for each of the opponent's n possible dice rolls. Let $\vec{X}_{-i}$ be a vector of length $D_{-i} + 1$, where the element $X_{-i}[j]$ ($0 \leq j \leq D_{-i}$) equals the probability of the opponent reaching h with exactly j dice showing face y or $\star$. $\vec{X}_{-i}$ is constructed in $O(n)$ time by iterating over each element of $\vec{\pi}_{-i}$, adding the probability to the appropriate entry of $\vec{X}_{-i}$ at each step. We can then compute the expected utility for player i with exactly j of his or her dice showing face y or $\star$. If player i called bluff, this expected utility is

$$U_i[j] = \sum_{\ell=0}^{x-j-1} (+1) \cdot X_{-i}[\ell] + \sum_{\ell=x-j}^{D_{-i}} (-1) \cdot X_{-i}[\ell];$$

if the opponent called bluff, the expected utility is $-U_i[j]$. Constructing $\vec{U}_i$ takes $O(n)$ time. Finally, we iterate over all $k \in \{1, ..., n\}$ and set $u_i[k] = U_i[x_k]$, where x_k is the number of dice showing face y or $\star$ in player i's kth private outcome. In total, the process takes $3O(n) = O(n)$ time.

3.3 Theoretical Analysis

CS, OPCS, SPCS, and PCS all belong to the MCCFR family of algorithms. As such, we can apply the general results for MCCFR to obtain a probabilistic bound on the average overall regret for CS and our new algorithms. Recall that in a two-player zero-sum game, minimizing average overall regret produces an ϵ-Nash equilibrium. The proof of Theorem 1 is in the appendix.

THEOREM 1. *For any $p \in (0, 1]$, when using CS, OPCS, SPCS, or PCS, with probability at least $1 - p$, the average overall regret for player i is bounded by*

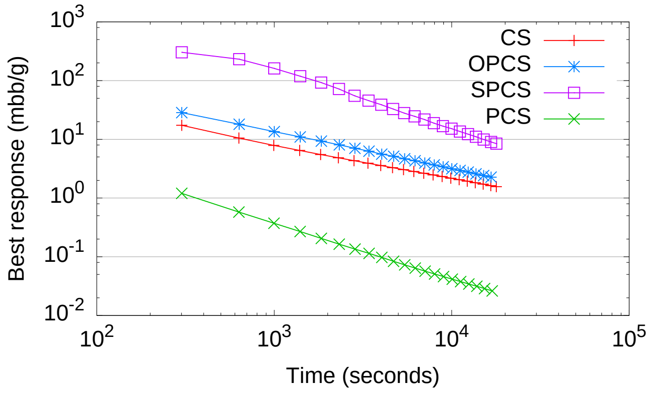
$$R_i^T \leq \left(1 + \frac{2}{\sqrt{p}}\right) \frac{\Delta_{u,i} M_i \sqrt{A_i}}{\sqrt{T}},$$

where M_i is a property of the game satisfying $\sqrt{|\mathcal{I}_i|} \leq M_i \leq |\mathcal{I}_i|$, $\Delta_{u,i} = max_{z,z'} |u_i(z) - u_i(z')|$, and $A_i = \max_{I \in \mathcal{I}_i} |A(I)|$.

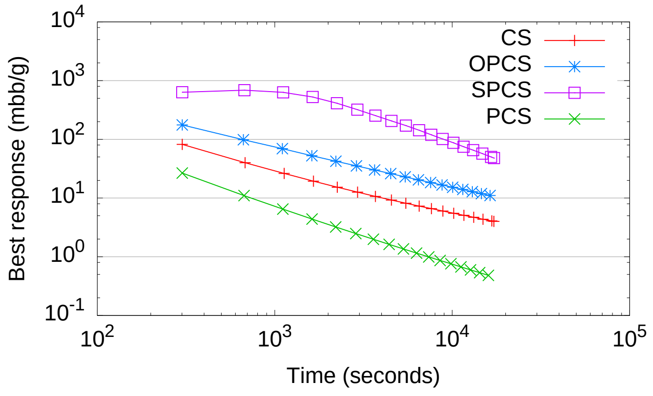
4. RESULTS

The efficacy of these new updates are examined through an empirical analysis in both poker and Bluff. We begin the analysis by examining the performance of CS, SPCS, OPCS and PCS in two small games, [2-1] hold'em and [2-4] hold'em. We will then present the performance of CS and PCS in a set of Texas hold'em abstract games, to investigate their usefulness under the conditions of the Annual Computer Poker Competition. Finally, we will apply CS and PCS to the Bluff domain.

Poker. [2-1] hold'em and [2-4] hold'em are games that are small enough to be tractably solved using all four of the CFR variants we are investigating: CS, SPCS, OPCS and PCS. As discussed in Section 3, SPCS, OPCS and PCS all perform $O(n)$ work at each



(a) $[2-1]$ hold'em, 16 million information sets



(b) $[2-4]$ hold'em, 94 million information sets

Figure 2: Log-log graphs displaying convergence of best response values over time for different CFR update methods in two small unabstracted hold'em like poker games. Best response values are in milli-big-blinds per game (mbb/g). Each curve shows the average performance over five independent runs.

terminal state, and are thus of comparable speed. However, all three require more time per iteration than CS, and to converge faster than CS, the advantage of each approach (more precise updates, more work per iteration, or both) must overcome this speed penalty.

Figure 2 shows the convergence of CS, OPCS, SPCS and PCS towards an optimal strategy in these small hold'em variants. We see that SPCS and OPCS converge slower than CS; the difference in speed is too great for the higher quality iterations. However, we find that PCS converges much more quickly than CS in these small games.

While [2-1] hold'em and [2-4] hold'em can be tractably solved using CFR, solving the much larger game of Texas hold'em is intractable. A common procedure used by competitors in the Annual Computer Poker Competition is to use a state-space abstraction technique to produce a smaller, similar game that can be tractably solved, and the resulting **abstract strategy** can then be used to select actions in the original game. The abstract strategy is an ϵ-Nash equilibrium in the abstract game, and we can measure its rate of convergence by calculating a best response within the abstract game. A critical choice in this procedure is the granularity of the abstraction. In practice, larger and finer-grained abstractions take longer to solve, but result in better approximations to a Nash equilibrium [4].

In Figure 3, we apply the CS and PCS algorithms to four sizes of abstract Texas hold'em games. The abstraction technique used in each is Percentile $E[HS^2]$, as described in [10], which merges information sets together if the chance events assign similar strength to a player's hand. An n-bucket abstraction branches the chance outcomes into n categories on each round.

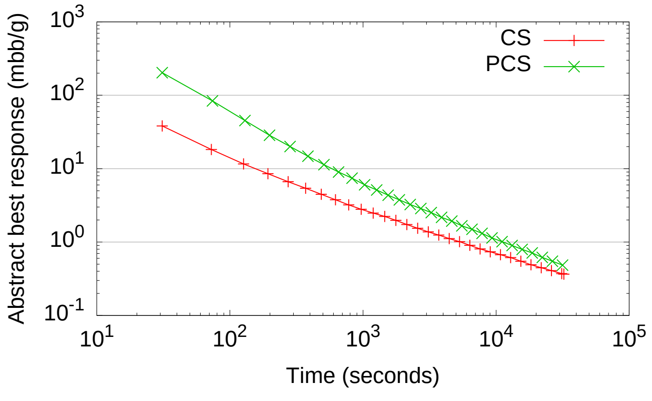
In the smallest abstract game in Figure 3a, we find that CS converges more quickly than PCS. As we increase the abstraction granularity through Figures 3b, 3c and 3d, however, we find that PCS matches and then surpasses CS in the rate of convergence. In each of these games, the chance sampling component samples outcomes in the real game and then maps this outcome to its abstract game equivalent. When a small abstraction is used, this means that many of the information sets being updated by PCS in one iteration will share the same bucket, and some of the benefit of updating many information sets at once is lost. In larger abstract games, this effect is diminished and PCS is of more use.

In the Annual Computer Poker Competition, many competitors submit entries that are the result of running CFR on very large abstract games. Computing a best response within such abstractions, as we did in Figure 3, is often infeasible (as many competitors use abstractions with imperfect recall). In these circumstances, we can instead evaluate a strategy based on its performance in actual games against a fixed opponent. We can use this approach to evaluate the strategies generated by CS and PCS at each time step, to investigate how PCS and CS compare in very large games.[1]

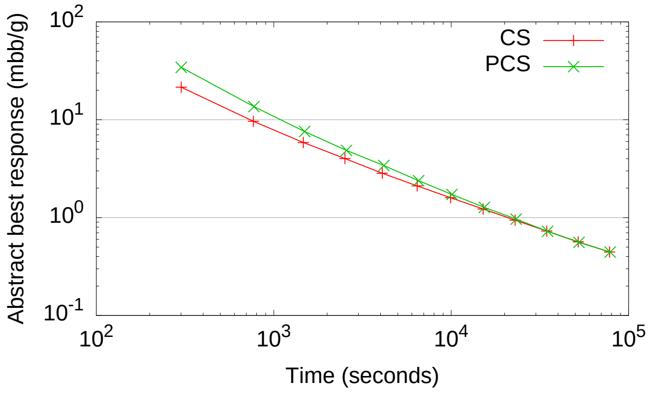
The results of this experiment are presented in Figure 4. The opponent in each match is Hyperborean 2010.IRO, which took third place in the 2010 Annual Computer Poker Competition's heads-up limit Texas hold'em instant runoff event. The y-axis shows the average performance in milli-big-blinds per game (mbb/g) over a 10-million hand match of duplicate poker, and the results are accurate to ± 1 mbb/g (so the difference in curves is statistically significant). The abstraction used for CS and PCS in this experiment uses imperfect recall and has 880 million information sets, and is similar to but slightly larger than Hyperborean's abstraction, which contains 798 million information sets. At each time step, the strategies produced by PCS perform better against Hyperborean than those produced by CS. Consider the horizontal difference between points on the curves, as this indicates the additional amount of time CS requires to achieve the same performance as PCS. As the competition's winner is decided based on one-on-one performance, this result suggests that PCS is an effective choice for creating competition strategies.

Bluff. Bluff(2,2) is small enough that no abstraction is required. Unlike poker, all of the dice rolls are private and there are no public chance events. In this domain, one iteration of PCS is equivalent to a full iteration of vanilla CFR (*i.e.*, no sampling). However, the reordering of the computation and the fast terminal node evaluation allows PCS to perform the iteration more efficiently than vanilla CFR. Figure 5 shows the convergence rates of CS and PCS in Bluff on a log-log scale. We notice that PCS converges towards equilibrium significantly faster than CS does. As noted earlier, PCS has two speed advantages: the fast terminal node evaluation, and the ability to reuse the opponent's probabilities of reaching an information set for many of our own updates. By comparison, vanilla

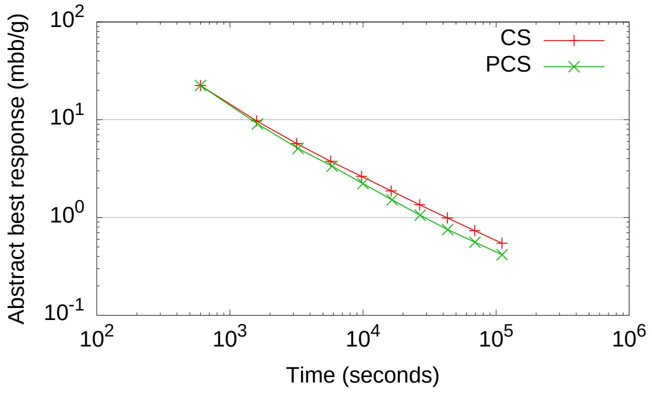
[1]Another possible evaluation metric is to compute the real-game exploitability of the strategies. However, the overfitting effect described in [4] makes the results unclear, as a strategy can become more exploitable in the real game as it approaches an equilibrium in the abstract game.



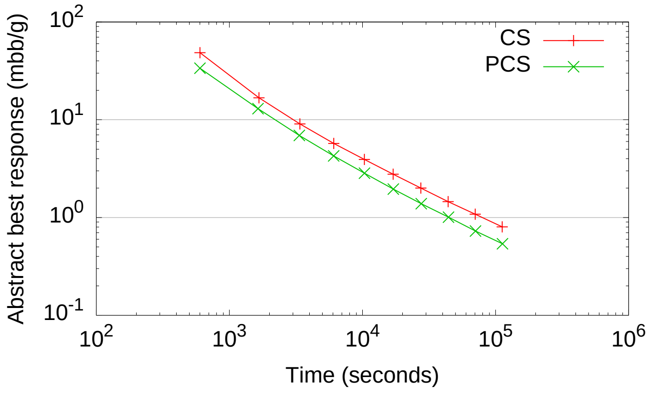
(a) 5 buckets, 3.6 million information sets



(b) 8 buckets, 23.6 million information sets



(c) 10 buckets, 57.3 million information sets



(d) 12 buckets, 118.6 million information sets

Figure 3: Log-log graphs displaying convergence of abstract best response values over time for different CFR update methods in two perfect recall abstractions of heads-up limit Texas hold'em poker. Best response values are in milli-big-blinds per game (mbb/g). Each curve shows the average performance over five independent runs.

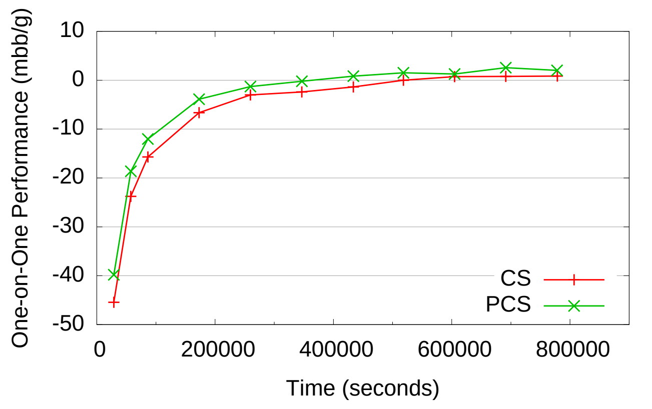


Figure 4: Performance of CS and PCS strategies in a large abstraction against a fixed, strong opponent.

CFR would traverse the action space 441 times to do the work of 1 PCS traversal. Similar to Figure 4 in the poker experiments, we can also compare the performance of CS and PCS strategies against a fixed opponent: an ϵ-Nash equilibrium for Bluff(2,2). This experiment is presented in Figure 6, and the fixed opponent is the final data point of the PCS line; the results are similar if the final CS data point is used. This result shows that PCS is also more efficient than CS at producing effective strategies for one-on-one matches.

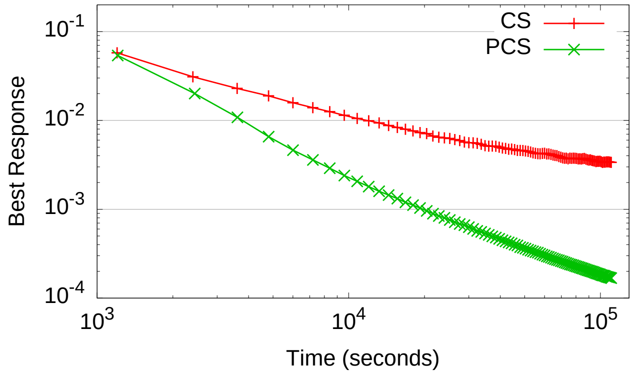


Figure 5: Log-log graph showing convergence of CS and PCS towards an equilibrium in Bluff(2,2). Each curve shows the average performance over five independent runs.

5. CONCLUSION

Chance Sampled CFR is a state-of-the-art iterative algorithm for approximating Nash equilibria in extensive form games. In this work, we presented three new CFR variants that perform less sampling than the standard approach. They perform slower but more efficient and precise iterations. We empirically demonstrated that Public Chance Sampling converges faster than Chance Sampling on

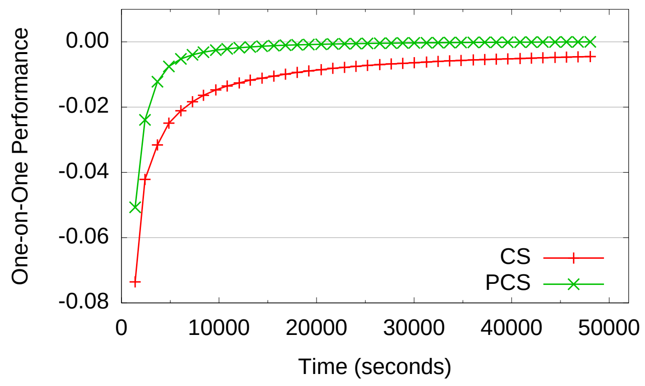


Figure 6: Performance of CS and PCS strategies against an ϵ-Nash equilibrium in Bluff(2,2)

large games, resulting in a more efficient equilibrium approximation algorithm demonstrated across multiple domains. Future work will look to tighten the theoretical bounds on the new algorithms to prove that they can outperform Chance Sampling.

Acknowledgments

The authors would like to thank the members of the Computer Poker Research Group at the University of Alberta for helpful conversations pertaining to this research. This research was supported by NSERC, Alberta Innovates Technology Futures, and the use of computing resources provided by WestGrid and Compute Canada.

6. REFERENCES

[1] A. Gilpin, T. Sandholm, and T. B. Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2007.

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

[3] S. Hoda, A. Gilpin, J. Peña, and T. Sandholm. Smoothing techniques for computing nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010.

[4] 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.

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

[6] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems 22 (NIPS)*, 2009.

[7] 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.

[8] M. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[9] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. Technical Report TR07-14, Department of Computing Science, University of Alberta, 2007.

[10] 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.

APPENDIX

Proof of Theorem 1. Let $\vec{a}_i$ be a subsequence of a history such that it contains only player i's actions in that history, and let $\vec{A}_i$ be the set of all such subsequences. Let $\mathcal{I}_i(\vec{a}_i)$ be the set of all information sets where player i's action sequence up to that information set is $\vec{a}_i$. Without loss of generality, assume $i = 1$. Let $\mathcal{D} = \mathcal{C}, \mathcal{O}_1 \cup \mathcal{P}, \mathcal{S}_1 \cup \mathcal{P}$, or $\mathcal{P}$ depending on whether we are using CS, OPCS, SPCS, or PCS respectively. The probability of sampling terminal history z is then

$$q(z) = \prod_{\substack{ha \sqsubseteq z \\ h \in \mathcal{D}}} f_c(a|h). \qquad (1)$$

Let $\vec{a}_i \in \vec{A}_i$, $B = \mathcal{I}_i(\vec{a}_i)$, and let $Q \in \mathcal{Q}$. By [7, Theorem 7], it suffices to show that

$$Y = \sum_{I \in B} \left(\sum_{z \in Z_I \cap Q} \pi_{-1}^{\sigma}(z[I]) \pi^{\sigma}(z[I], z) / q(z) \right)^2 \leq 1.$$

By (1) and definition of π_{-i}^{σ}, we have

$$Y = \sum_{I \in B} \left(\sum_{z \in Z_I \cap Q} \pi_2^{\sigma}(z[I]) \pi_{1,2}^{\sigma}(z[I], z) \prod_{\substack{ha \sqsubseteq z \\ h \in \mathcal{C} \setminus \mathcal{D}}} f_c(a|h) \right)^2. \qquad (2)$$

Now by the definition of $\mathcal{Q}$, for each $h \in \mathcal{D}$, there exists a unique $a_h^* \in A(h)$ such that if $z \in Q$ and $h \sqsubseteq z$, then $ha_h^* \sqsubseteq z$. Next, we define a new probability distribution on chance events according to

$$\hat{f}_c(a|h) = \begin{cases} 1 & \text{if } h \in \mathcal{D}, a = a_h^* \\ 0 & \text{if } h \in \mathcal{D}, a \neq a_h^* \\ f_c(a|h) & \text{if } h \in \mathcal{C} \setminus \mathcal{D}. \end{cases}$$

Notice that $\prod_{ha \sqsubseteq z, h \in \mathcal{D}} \hat{f}_c(a|h)$ is 1 if $z \in Q$ and is 0 if $z \notin Q$. Thus from (2), we have

$$\begin{aligned} Y &= \sum_{I \in B} \left(\sum_{z \in Z_I} \pi_2^{\sigma}(z[I]) \pi_{1,2}^{\sigma}(z[I], z) \prod_{\substack{ha \sqsubseteq z \\ h \in \mathcal{C}}} \hat{f}_c(a|h) \right)^2 \\ &= \sum_{I \in B} \left(\sum_{z \in Z_I} \hat{\pi}_{-1}^{\sigma}(z[I]) \hat{\pi}^{\sigma}(z[I], z) \right)^2 \\ &\quad \text{where } \hat{\pi}^{\sigma} \text{ is } \pi^{\sigma} \text{ except } f_c \text{ is replaced by } \hat{f}_c \\ &= \sum_{I \in B} \left(\sum_{h \in I} \hat{\pi}_{-1}^{\sigma}(h) \sum_{z \in Z_I} \hat{\pi}^{\sigma}(h, z) \right)^2 \\ &= \sum_{I \in B} \left(\sum_{h \in I} \hat{\pi}_{-1}^{\sigma}(h) \right)^2 \leq 1, \end{aligned}$$

where the last inequality follows by [7, Lemma 16]. □.