Survey of:
Safe Learning in Zero-sum Games by Kevin Waugh and Michael Bowling

"Most of the money you'll win at poker comes not from the brilliance of your own play, but from the ineptitude of your opponents." – Lou Krieger

1 Introduction

Poker is a game involving multiple players (or agents), a mix of known and unknown information, various payoff and betting structures and many possible strategies. Whilst there are some simple probabilistic elements to the game, many poker variants have resisted definitive analysis. Recent competitions which focus on the performance of automated poker playing systems (such as the AAAI Computer Poker Competition) have generated large amounts of academic interest and research into the analysis and theory of this class of game.

The paper surveyed examines two new approaches to generating strategies in poker, which both combine some element of online learning with precomputed strategies. In this context online learning means improving ones strategy as each round (or hand) of a game of poker is played, using the observed outcome of each hand as feedback to the learner. In this way the authors hope to retain some worst-case safety bounds associated with the precomputed strategies, alongside the ability to exploit possible weaknesses in opponents strategies which may have been observed over the course of several rounds of the game.

This survey is organized as follows. In the "Framework" section we review the framework within which much of this analysis takes place, namely the extensive game framework. We also define several of the terms and concepts that are used in the remainder of the paper. In the "Previous Approaches" section we discuss some of the previous results and approaches referenced by the surveyed paper, and the motivation behind the approach taken by the papers authors. In the "Evaluation" section we define the evaluation framework employed by the authors to access the effectiveness of their new approaches, and the motivation behind this approach. In the "ϵ-Safe Learning" and "Equilibrium Restricted Learning" sections we discuss these two key results of the paper. Finally in the "Conclusions" section we provide some conclusions and final thoughts.

2 Framework

The extensive game model is employed across a number of game theory domains, and is a general approach to modeling games which consist of some number of players, possible sequential actions of players at certain points in the game, and payoffs associated with the end of a given game.

In this paper, the authors restricted their view to finite, zero-sum, two player games with imperfect information and perfect recall, which is reflected in the following descriptions, although it is worth noting that all of these concepts can be generalized beyond these restrictions.

2.1 Extensive Game Model

The extensive game model allows us to represent many classes of games which involve multiple agents (players), including common games such as chess, backgammon and poker. An extensive game is a tree where each internal (non-leaf) node corresponds to a decision point for a specific player (where we take the special case player "nature" to be the player that is used to perform randomized actions where needed), and edges correspond to possible choices or actions made at a decision node by the player associated with that node. Terminal nodes (leaves) of the tree represent a possible endings of a game, and are associated with a payoff for each player. This allows us to represent the sequential nature of a game where players take turns making decisions (interlaced with decisions taken by our special case player "nature" as required).

Formally we can say that an extensive form game for two players, consists of the following:

- A set of possible action sequences, H. For any possible sequence of actions that can occur in our game, we should have the corresponding sequence in H. This should include the empty sequence $\{\}$ (e.g. no actions have taken place) and a special subset of sequences that correspond to each terminal node in our tree, called $Z \subseteq H$. From this definition it follows that for any $h \in H$ where h s not equal to the empty sequence, any prefix of h must also be in H. We can think of each history (except for those in Z) as representing a decision node.

- The set of actions available after the sequence of actions $h \in H$ is $A(h) = \{a : (h, a) \in H\}$. In other words it is the set of actions we can take which leads to a legal new sequence of actions. Note that for any $z \in Z \subseteq H$, $A(z) = \{\}$ since there are no possible actions we can take from a terminal node.

- A function P which associates every $h \in H \backslash Z$ (e.g. non-terminal histories) with a player. This represents the player whose turn it is to chose an action after a given history. Note that in addition to the two players playing the game, we have a special case player "nature" which represents the situation when the next action is to be chosen randomly and independently of any other decisions in the game. For example in the case of poker, we would assign the initial actions of dealing random cards to nature, rather than any of our actual players.

- For each player we define a utility function that maps a terminal history to a payoff for that player. So for any $z \in Z$, $u_1(z)$ is the utility function for player 1, and similarly for player 2 we have u_2. Since we have a zero-sum game we know that for any $z \in Z$, $u_1(z) = -u_2(z)$. Alternatively we can think of this as being the constraint that the payoff for all players must sum to zero for each terminal history.

The paper focuses on a specific class of game, which characterizes poker, which is that of an imperfect information game. Due to imperfect information, it may not be possible for a player to distinguish between different histories. As a clear example, a player cannot distinguish between histories which are identical except for the actions taken by nature when dealing cards to his opponent (assuming that the player cannot see his opponents cards). This concept is formalized through the idea of information sets.

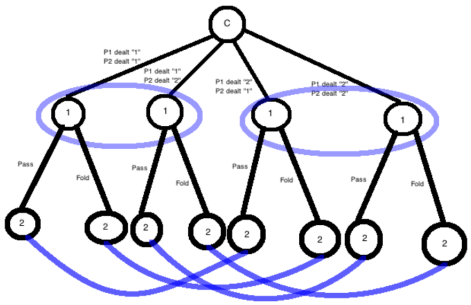
Each information set is associated with a given player and is a set of histories (or alternatively decision nodes) which that player cannot distinguish between. For our purposes this has the consequence that any strategy for the associated player, must have identical behavior across all histories in a information set (since the player cannot tell which of these histories he is actually playing from). We note that this further implies that the set of available actions (or edges) from each decision node within a given information set must be identical (as implied by the definition of information sets above). Information sets partition H, and are defined as equivalence classes where:

- For each player i, $\mathcal{I}_i$ is a partition of $\{h \in \text{H: } P(h) = i\}$ and for all $I \in \mathcal{I}_i$, for all $q, r \in I$ we must have that the same set of actions can follow q and r, so $A(q) = A(r)$ and that player i cannot distinguish in any way between q and r. We call $\mathcal{I}_i$ the information partition for player i and I_i an information set for player i where $I_i \in \mathcal{I}_i$.

We note that perfect recall further refines the definition of information sets by ensuring that since all players are assumed to have perfect recall (in the sense that they can recall all actions that they have taken so far in the game tree), histories within an information set must be identical on those actions within each sequeunce which correspond to the player associated with the particular information set, since the player could otherwise distinguish between such histories by recalling his previous actions.

To view this as a tree you can take the set of $H \backslash Z$ as representing decision nodes, and have an edge between nodes h and h' if there is a action $a \in A(h)$ such that $(h, a) = h'$. I have given a simple example of a tree for a game where players are dealt one of two possible cards by nature (where we assume that we each player is dealt from a separate deck so cannot imply the value of his opponents card from his own card), and then player 1 either passes

or folds. Each circle represents a decision node, and is marked with the associated player (using c for nature). Information sets for player 1 are shown as blue ovals capturing the nodes in each of player 1's two decision sets, and information sets for player 2 are depicted with blue lines linking nodes in each of its four decision sets. These represent the idea that player 1 cannot distinguish between the histories which correspond to player 2 being dealt either card 1 or 2, and player 2 cannot distinguish between the histories corresponding to player 1 being dealt either card 1 or 2. The picture represents the start of a game tree, not an entire game tree, so for example player 2's possible actions are not shown, nor do we show terminal nodes.



2.2 Strategies and Equilibrium

Two basic definitions related to this framework are that of a strategy and a strategy profile. A strategy for a given player i, usually denoted σ_i defines a decision making process for player i that can be used to determine player i's actions at any decision node for player i in our tree. We can think of this as being a function that, for every information set corresponding to player i, defines a probability distribution over possible actions available to player i at nodes within the information set. So $\sigma_i(I, a)$ is the probability that strategy σ_i assigns to taking action a given that player i is at a decision node in information set I.

A strategy profile is a set of strategies, one for each player. So in our case of two player games it is a pair of strategies $\sigma = \{\sigma_1, \sigma_2\}$ denoting that player i follows strategy σ_i. It is worth noting that strictly speaking strategies as defined here are considered mixed strategies in the sense that they are a probability distribution over pure strategies, where a pure strategy is defined as a deterministic strategy (i.e. a specific choice of which action to take at each decision set). Clearly mixed strategies generalize pure strategies (in the sense that every pure strategy is a mixed strategy where the mixed strategy places its whole distribution on the corresponding pure strategy) and are appropriate to an adversarial setting such as a two-player zero-sum game.

Given the notion of a strategy and utility function, we can now define two related concepts, that of a equilibrium and a best response. An equilibrium is a strategy profile, such that if any player were to unilaterally choose an alternative strategy they would reduce their expected utility. We use the term equilibrium strategy to denote a strategy from the equilibrium associated with the specific player. A best response for a given player i, is a strategy for player i which, given knowledge of the strategies being employed by the other players, maximizes the expected utility for player i. In the case of an equilibrium it is clear that if player 1 plays an equilibrium strategy then the best that player 2 can do is play the associated equilibrium strategy, and vice versa. We can formally define these as follows:

Let Σ_i be the set of all possible strategies that player i can employ.

Let $\pi^\sigma(h)$ be the probability of reaching history $h \in H$ if players choose actions according to the strategy profile σ.

Then the expected utility of player 1 under this strategy profile is $u_1(\sigma) = \sum_{h \in Z} u_i(h) * \pi^\sigma(h)$

Then the set of best response strategies for player 1, given that player 2 is playing using strategy σ_2 is $BR(\sigma_2) \subseteq \Sigma_1$ where:

$BR(\sigma_2) = \arg\max_{\sigma_1 \in \Sigma_1} u_1(\sigma_1, \sigma_2)$

If we have a situation where each player is playing a best response strategy, given the strategy of the other player, we have an equilibrium. We note that such an equilibrium will not necessarily be unique, as there may be several such combinations of strategies that fulfill this criteria. If we have such an equilibrium, then it follows that for any individual player, there is no gain from changing its individual strategy, since from the above this cannot increase its expected utility given the other players strategy.

A related concept used extensively in this paper is that of an ϵ-equilibrium. This is similar to the above, but weaker in the sense that if a strategy profile is a ϵ-equilibrium, then a player can increase their utility by at most ϵ through a unilateral change in his strategy. Clearly, for $\epsilon = 0$ we have that an ϵ-equilibrium is equivalent to an equilibrium. We can formally define both of these concepts as follows:

An equilibrium is a strategy profile σ where:

$u_1(\sigma) \geq \max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2)$ and

$u_2(\sigma) \geq \max_{\sigma_2' \in \Sigma_2} u_1(\sigma_1, \sigma_2')$

An ϵ-equilibrium is a strategy profile σ where:

$u_1(\sigma) + \epsilon \max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2)$ and

$u_2(\sigma) + \epsilon \geq \max_{\sigma_2' \in \Sigma_2} u_1(\sigma_1, \sigma_2')$

The authors of the paper note that in a zero-sum context, any equilibrium strategy has a minimax guarantee. This can be thought of as an equilibrium strategy minimizing the

maximum utility of our opponent, which, since we are in a zero-sum context, means we are maximizing our minimum utility. It is worth further noting that any finite game has at least one equilibrium.

2.3 Exploitability

One measure of the performance of a particular strategy is its exploitability. If we let v_1 be the value of the game to player 1 under an equilibrium strategy (e.g. the expected utility for player 1 under any equilibrium strategy), then we define exploitability of a strategy as:

$ex(\sigma_1) = \max_{\sigma_2 \in \Sigma_2}(v_1 - u_1(\sigma_1, \sigma_2))$

We define exploitability for player 2 in a similar fashion.

We note that this is well defined, since in any zero-sum extensive game, for any two equilibrium (σ_1, σ_2) and (σ'_1, σ'_2) we have that $u_1(\sigma_1, \sigma_2) = u_1(\sigma'_1, \sigma'_2)$ and that both (σ_1, σ'_2) and (σ'_1, σ_2) are also equilibrium.

A strategy σ_1 such that $ex(\sigma_1) \leq \epsilon$ is known as an ϵ-safe strategy, and is a member of an ϵ-equilibrium as defined above.

2.4 Abstraction

Equilibrium for finite, two player, imperfect information games with perfect recall can be found using a linear programming approach, usually dependent on a slightly different representation called the realization plan representation. Whilst this representation is linear in the number of game states (histories), for large games, this may not be tractable. For example in heads-up (two player) limit hold'em there are 10^{18} game states, which lead to intractable systems of linear equations. The common approach taken is to instead solve for abstractions of the game model. Having solved for an abstraction of a game model to find an equilibrium strategy that holds in the abstraction, we then apply that strategy in the original game model with appropriate transformations as required (to transform our abstracted strategy back to a strategy that makes sense in the original game model).

Abstraction techniques involve reducing the number of information sets for a given player, and hence usually involve merging information sets, intuitively meaning that the strategy will no longer differentiate between decisions made in these merged information sets. This can be done in a number of ways such as bucketing different card values, limiting the types of betting possible in different rounds or focusing only on different classes of outcome (for example having a pair of cards vs. not having a pair of cards). The assumption usually made is that the more granular the abstraction used (i.e. the closer it is in some sense to

the original model), the better the equilibrium strategy from the abstraction will perform in the original game model.

It is worth noting however that there is no formal theory behind this assumption [1], and in fact in some pathological cases this has been shown not to hold. For clarity we also note here that when abstracting a game model it can be done from either players perspective, using potentially different abstractions for different players. This may come about if we wish to model our opponents strategy in a certain tractable abstraction, but model our own strategy in a more granular abstraction for which we have several precomputed strategies.

3 Previous Approaches

Two specific previous approaches are mentioned in this paper, that of a data driven approach, and that of an online learning approach. Both of these approaches try to balance the use of an equilibrium strategy with that of a best response strategy tailored to a specific opponent strategy. Both of the approaches outlined below cannot tailor a strategy to specifically exploit weaknesses of an opponent during live game play, which motivates the authors new approaches. The authors approach is motivated by the observation that whilst playing an arbitrary equilibrium strategy has a comforting worst case minimax guarantee, it may be failing to exploit possible weaknesses in an opponent given that the opponent is almost certainly not playing an exact equilibrium strategy, and should thus be exploitable.

3.1 Data Driven Approach

In the data driven approach, we first observe our opponent playing for a number of hands. From these observations we construct a model of the strategy employed by our opponent (within some suitable tractable abstraction). One option now would be to play a Frequentist Best Response which is a strategy from the set of best responses where we assume our opponent is playing the strategy exactly implied by our model (where our model is simply a maximum likelihood model based on our observations of our opponents strategy). However this may perform poorly if for example our model is a poor approximation of our opponent (either due to a low number of observations, or because our opponent is not actually playing with a static strategy which our model assumes). It will also be very brittle in the sense that against any other opponent, playing the strategy which has been entirely optimized based on one opponent, will probably perform poorly (unless the new opponents strategy is coincidently close to our model).

To account for the idea that the observations may not precisely define the opponents strategy, and to try and make the model more robust, we actually assume our opponent is

playing a mixed strategy which, with probability p matches the model we have observed, and with probability $(1-p)$ it plays with some other arbitrary strategy from its set of possible strategies. We have now constructed a new set of possible strategies for our opponent which we call $\Sigma_2^{p,\sigma_{FIX}}$ consisting of the set of mixed strategies of σ_{FIX} (the strategy implied by our model) and some other strategy in Σ_2. We can now solve this perturbed game to find the set of equilibrium, which should be more robust against an opponent whose strategy is not exactly modeled by our observations. With low values of p we also make our strategy more robust against opponents whose strategies may differ significantly from that implied by our model.

We let a restricted Nash equilibrium be a pair of strategies (σ'_1, σ'_2) where $\sigma'_2 \in BR^{p,\sigma_{FIX}}(\sigma'_1)$ and $\sigma'_1 \in BR(\sigma'_2)$. This is effectively saying that given a restricted set of possible strategies for player 2 (restricted to the set of mixed strategies where it plays σ_{FIX} for p percent of the time), each player is playing a best response. So the approach taken is to solve for this restricted Nash equilibrium, and play using the associated strategy for player 1.

Unfortunately this approach is somewhat unfeasible in live play as it requires many observations of our opponents play (ideally where our opponent is being observed playing against an opponent which forces it to make decisions that improve our model, i.e. which are well distributed over the opponents decision nodes). As an example in Texas Hold-em poker, 10,000 hand observations were required to improve the performance over a static equilibrium strategy. In addition since we must solve the perturbed game in order to define our strategy, this is computationally intensive (it may take several days / weeks depending on the abstractions and method being employed) and so we cannot employ this technique in an iterative way during actual game play (i.e. by refining our model / observations and recomputing a best response).

3.2 Online Learning

In the online learning approach we precompute a set of possible strategies (which could for example be best responses based on a sample of possible opponent playing strategies that we think our opponent may employ), and using the approach of a multi-arm bandit algorithm select from these strategies as we play (where each strategy in our set of precomputed strategies is represented by a bandit arm, and our algorithm chooses amongst this set using the observed outcome of each choice to refine its decision process over time). This approach has performed well in competition and generalized to other areas (such as robot soccer). However it has the draw back that we must select from a predefined set of strategies which may not include a strategy that can exploit our opponent to a large degree, since our set of strategies is static. In addition it is hard to make worst case guarantees about the performance of this

algorithm without assuming some properties of the underlying precomputed strategies.

4 Evaluation

The variant of poker used for evaluation is Leduc Hold'em. For a precise description of the game see the appendix but in summary it is a non-trivial poker variant with a strategy space of 193 dimensions for each player. This variant was chosen as it is tractable for computing best response strategies, solving for equilibrium and employing polynomial methods such as linear programming. As a test bed for evaluating new strategy determination methods it is therefore possible to give precise numeric results of match outcomes, computed exactly rather than by simulation as would be required in a less tractable variant of poker, such as that used in the aforementioned poker competitions.

In order to evaluate the success or otherwise of a new poker strategy we must have some opponents against which we can measure this success. The approach taken by the authors was to generate a number of abstractions of Leduc Hold'em, and then solve within those abstractions to generate an equilibrium strategy that holds for each abstraction. Five abstraction methods were used, which generates 25 equilibrium strategies, one for each pair of abstraction methods (since in a two player game, each player can be abstracted in one of 5 ways). The abstraction techniques at a high-level were to put dealt cards into smaller classes (so for example treating a King or Queen equally) and to only consider whether the community card dealt either paired, or didn't pair our dealt card. This opponent generating technique was used as it is a approximation of the real world in the sense that strategies in larger games are usually generated in this manner by solving to find equilibrium in game abstractions and then using these strategies in the unabstracted game model.

In the actual aforementioned competition, each opponent plays each opponent for 3,000 hands to generate an expected outcome, but since in Leduc Hold'em, it is possible to exactly calculate expected outcomes, this is the approach taken when evaluating new strategies. The best strategy of the 25 generated strategies wins at 83.4 millibets per hand (mb/h) on average against all opponents, and the worst loses at a rate of 114.1 mb/h. It is worth noting that the exact arbitrarily chosen equilibrium strategy (i.e. that which is computed in the unabstracted Leduc Hold'em game model) is not the best performer, having a win-rate of 59.9 mb/h. The intuition here is that whilst it never loses against any opponent (as expected) it also fails to exploit any opponent in an optimal way, in comparison to the best abstracted strategies which do happen to exploit some of the weaker abstracted strategies far more.

The authors note that there exists a static equilibrium strategy which wins at 161.6 mb/h, and so if we had behavior which mimics the ability to choose optimally amongst our set of

possible equilibrium strategies we ought to be able to increase our performance significantly. Furthermore the best dynamic strategy (which plays a best response strategy tailored to the specific opponents strategy) has a win-rate of 308.3 mb/h, so if we were able to dynamically determine an appropriate strategy we may be able to improve performance even further.

4.1 Regret

It is worth digressing a little to explore the concept of regret as it applies in this context. Regret minimization is the approach where at each iteration of an algorithm we attempt to minimize our regret, where regret is defined as the difference between an optimal strategy chosen in hindsight, and the actual strategy played. Formally we have that the average overall regret of player 1 at round T is:

$R_1^T = \frac{1}{T} \max_{\sigma_1' \in \Sigma_1} \sum_{t=1}^{T} (u_i(\sigma_1', \sigma_2^t) - u_i(\sigma^t))$ where
σ_i^t is the strategy played by player i on round t, and $\sigma^t = (\sigma_1^t, \sigma_2^t)$
We define R_2^T in a similar fashion.

This can be interpreted as saying that the regret of player 1 is the difference between the expected utility of the optimal fixed strategy for player 1 played against each of player 2's strategies in each round, and the expected utility of the actual chosen strategy of player 1 against the same set of player 2's strategies in each round. If we imagine that that the strategy for player 2 is chosen in an adversarial context in each round, then this has the slightly unintuitive nature that we are comparing our actual performance against an adversary (which we can imagine choosing a new strategy for player 2 in each round based on previous performance of its choices against our choices) with our optimal performance against the adversary which is forced to choose strategies based on our initial choices.

We note that there is a well established theory that states, that if both player's average overall regret is less than ϵ at round T, then the strategy profile at round T is a 2ϵ-equilibrium. This leads to a commonly used alternative (to linear programming) approach to equilibrium solving known as self-play, where if we play regret minimizing algorithms against themselves, then as t goes to infinity (and hence average overall regret tends to zero) we are effectively computing an equilibrium.

This notion is further refined by the concept of Counterfactual Regret. The idea here is that rather than minimizing overall average regret, we instead split out regret into additive regret terms which can be independently minimized, and show that minimizing each of these terms leads to overall average regret minimization. Hence if we have an algorithm that minimizes counterfactual regret, we can also compute equilibrium [2]. Counterfactual regret is defined on information sets, and is the analogue of regret, but instead of looking at the

difference between optimal and chosen strategies across an entire game, we look at this difference as it would apply to regret at a particular information set. The counterfactual regret at a specific information set can therefore be considered to be the players regret in its decisions made at a particular information set in terms of the counterfactual utility of that information set which is defined as being the expected utility given that the information set is reached and all players play using a given strategy profile except player i who plays to reach the specified information set.

One further recent innovation related to the concept of counterfactual regret is the observation by [3] that it is sufficient, when calculating expected counterfactual regret, to use a sample of possible outcomes (terminal histories) to generate an estimate of counterfactual regret, rather than considering all possible outcomes stemming from an information set. This can greatly improve the time required to compute equilibrium using this approach as the efficiency of sampling a small set of possible outcomes outweighs the loss of accuracy associated with this approach (in other words we may need more iterations with this sampling based approach, but the overall time per iteration is greatly reduced). A number of sampling techniques have been considered, but the one that is appropriate in an online learning sense is that of outcome sampling, which is the idea that we only sample one outcome of a strategy at a particular information set, which is the outcome that corresponds to our observed payoff during play.

So to summarize if we have an online learning algorithm which chooses between possible actions at information sets independently, by using the observed outcome of each hand following a particular set of action choices, it is possible to minimize counterfactual regret at each information set, and hence overall average regret, and therefore converge to an equilibrium strategy. This is exactly the approach taken below.

4.2 Baseline

To establish a baseline for current online learning methods within the evaluation framework, the authors assessed the performance of the following online learning based algorithm within the framework. The online algorithm treats each information set like an instance of the multi-armed bandit problem. It therefore sees each of the possible actions it can take at a given information set as corresponding to pulling one of the bandits arms, and tries to learn, over multiple rounds, what distribution of probabilities over each of these arms is optimal in terms of minimizing regret at that information set. This algorithm can be thought of as an extension of the familiar randomized weighted majority algorithm. At each information set we have one expert per possible action, and that expert always tells us to take the associated action. We want to learn a probability distribution over our experts advice such that this

probability distribution minimizes regret.

The authors used the Exp3 algoritm of [4] with the GL algorithm of [5] as the underlying learner. Specifically they used this pairing at each information set, so we actually independently learn at each information set (although every algorithm will share the same observed result). We first examine the GL algorithm and then look at its use as a sub-routine of the Exp3 algorithm.

GL Algorithm over K actions

Input: parameter α

Initialization: Set initial weight w_i of action $a_i$to $\frac{1}{K}$ for all $i \in \{1, ..., K\}$

Repeat for $t = 1, 2, ...$ until game ends:

1. Choose action i_t according to the distribution $P(t)$ where $P_i(t) = \frac{w_i^t}{\sum_{j=1}^{K} w_j^t}$ and w_i^t is the weight associated with action i at the beginning of iteration t.

2. Observe the reward vector $x(t)$ and receive payoff $x_{i_t}(t)$.

3. Update the weights:

   If an action i receives a payoff greater or equal to zero then set:

   $w_i^{t+1} = w_i^t * (1 + \alpha)_i^g(t)$ where $g_i(t)$ is the observed gain of action i.

   If an action i receives a payoff less than zero then set:

   $w_i^{t+1} = w_i^t * (1 - \alpha)_i^l(t)$ where $k_i(t)$ is the observed loss of action i.

We first note that GL here stands for Gain / Loss to indicate that it deals with observed rewards with can be either positive or negative as above. We also note that in the execution of the algorithm we expect to receive the full reward vector $x(t)$ which would correspond to the reward associated with every possible choice we could have made, and thus we are able to update the weights of all of our actions in an unbiased way on each iteration. The α variable controls how aggressively the learner updates its weights on each iteration which can be thought of a measure of how aggressively we want to deviate from our uniform distribution, or equally how much emphasis we want to put on each individual observed reward. We also note that since the algorithm operates at each information set (wrapped by the Exp3 algorithm as below) it can be seen to be minimizing counterfactual regret at each information set, and thus the win-rate of this algorithm will approach that of the best dynamic strategy over time.

As noted above this algorithm expects to learn in a full information environment, which is not possible when playing poker in an online setting, since we actually only get to observe

the gain or loss associated with our specific choice. Given this situation, it is clear that we cannot simply update the weight associated with our specific choice and then choose based on these weights as the update will be biased in the sense that it will only update one of our action weights. Additionally the algorithm will fail to fully explore all possible actions and be biased (either towards or against) those actions which are initially chosen at random. To overcome these difficulties the algorithm is wrapped in the Exp3 algorithm. This algorithm takes a parameter γ which it uses to generate a new probability distribution over actions which is a linear interpolation between the probability distribution maintained by the GL algorithm and the uniform distribution (linearly combined using γ). Furthermore, given an observed gain or loss of a chosen outcome, it generates an unbiased estimate of this for the GL algorithm by dividing the gain or loss by the probability with which we selected the action.

Exp3 Algorithm

Input: parameter η and parameter $\gamma \in (0, 1]$

Initialization: Initialize underlying GL algorithm

Repeat for $t = 1, 2, ...$ until game ends:

1. Get the distribution $p(t)$ from GL.
2. Select action i_t to be j with probability $\hat{p}_j(t) = (1 - \gamma)p_j(t) + \frac{\gamma}{K}$.
3. Observe the reward associated with this action $x_{i_t}(t)$.
4. Generate an unbiased estimator of the whole reward vector $\hat{x}(t)$ defined by:

   $< 0, 0, ..., 0, \frac{x_{i_t}(t)}{\hat{p}_{i_t}(t)}, 0, ..., 0, 0 >$

   In other words it uses 0 for every point in the vector that doesn't correspond to action i, and an unbiased estimate otherwise.

The performance of this approach is noted by the authors to be poor over a small number of rounds. It is not until 56,000 hands are played that the online learner has a positive win-rate. It begins to win the tournament after 106,000 hands are played. This performance seems reasonable as, given the above set up, it is clear that the online learner has to effectively learn the game from scratch, with no preconceived notions as to what constitutes good play. In this sense it has to have a reasonable sample of the whole strategy space before it can begin to exhibit decent performance, and given the size of the strategy space in these types of game, that is prohibitive.

4.3 Motivation

It is clear from the above that a pure online learner performs poorly in any realistic environment. Conversely previous attempts to marry online learning with precomputed strategies exhibit a number of failings either being impractical in a realistic environment due to computation overheads, or not being flexible enough. This motivates the approach taken by the authors which attempts to perform online learning over restricted strategy spaces, where the restriction on the strategy space allows us to make worst-case guarantees whilst the online learning allows us to exploit possibly weak opponents.

5 ϵ-Safe Learning

For games such as poker, there are several possible equilibrium strategies, each with the same minimax guarantee. ϵ-safe learning is motivated by the idea that although playing an arbitrary equilibrium strategy has a minimax guarantee, if we assume that our opponent is not necessarily playing an equilibrium strategy, then intelligently selecting our choice of equilibrium strategy may allow us to exploit our opponent, whilst at the same time maintaining a minimax guarantee.

Since in general poker games are only solvable in abstraction (i.e. solving the unabstracted game is intractable as there are many game states), it is almost certain that in fact our opponent will not be playing an equilibrium strategy in the unabstracted model, so it seems reasonably to expect that by choosing a specific equilibrium strategy to play we will often outperform an arbitrarily chosen strategy. As well as considering the strategy space of equilibrium strategies, the authors consider the more general case of strategy spaces consisting of ϵ-safe strategies, as defined above. In this case we clearly have a worse worst-case guarantee (our expected regret may be ϵ less than the minimax value), but equally we have a wider choice of strategies that our learner may employ, and hence increase the likelihood of finding a strategy that may be able to better exploit an opponent.

The authors observe this within their framework by noting that if they manually choose the best static equilibrium strategy to play against our suite of opponents, its win-rate of 65.8 mb/h is almost 10% higher than the previously arbitrarily chosen static equilibrium strategy. If they are allowed to choose the best static strategy per opponent, then this win-rate increases even further to 67 mb/h. They further note that if they relax the minimax guarantee and allow themselves to manually choose from amongst the ϵ-safe equilibrium strategies then with a loss of less than 0.005% of the minimax guarantee the best dynamically chosen strategy (where we allow ourselves to choose a new strategy per opponent) now places first overall in our competition.

At a high level the ϵ-safe learning algorithm employed by the authors, for a given ϵ, selects from amongst the set of all ϵ-safe equilibrium strategies on each round of the game, using online observations of regret for previous choices to bias its current choice. The algorithm can be broken down into two modes of operation. An exploit mode where it uses its previous observations to choose the best strategy amongst its strategy set consisting of ϵ-safe strategies, and an explore mode where it tries out other options in order to try and learn a better model for making future best choice predictions.

5.1 Realization Plan Model

We assume that our opponent is playing a fixed strategy, and instead of choosing between strategies we actually choose between realization plans. A realization plan for a given player is a representation of a strategy consisting of a function which maps any sequence of actions for the player to a weight. We can then think of a strategy as being a choice from amongst those sequences which are possible at each information set, where our choice is weighted using the realization plan function. The advantage of this approach is that it is possible to calculate the expected reward of a realization plan as a linear function of our realization plan, assuming we have a fixed adversary, and so we are able to leverage the optimization discussed below in the Explore subsection. Furthermore this also allows us to construct an efficient optimization oracle required for the Exploit subsection below.

Formally we say that any decision node in a tree associated with player i induces a sequence which is the set of actions taken by player i to arrive in that node. Due to the perfect recall property we are thus guaranteed that for any information set, we have at most one sequence of actions that can lead to any node in that set. The sequence associated with information set q is denoted μ_q. It can be seen that each possible action a at q will form a new unique sequence μ_q, a, and that this can be used to specify any possible sequence (excepting the empty sequence). We say that the set of sequences for player i is $|S_i|$.

A realization plan x of a mixed strategy of player 1 can be thought of as a non-negative vector of length $|S_1|$ which maps each sequence to a realization probability and from the above can be characterized as:

$x(\{\}) = 1$, $\sum_{a \in A(h)} x(\mu_h, a) = x(\mu_h)$ for all $h \in H_1$

and similarly we say y is the realization plan for player 2.

In other words the total weight on associated with the set of sequences leading out of a node must be equal to the weight of the sequence leading to a node. A realization plan in this form can be thought of as encoding a strategy whose choice at a decision node h is to choose action a with probability $\frac{x(\mu_h, a)}{x(\mu_h)}$ if $x(\mu_h)$ is non-zero, and with probability 0 otherwise.

We can consider the above constraints as being encoded using matrices E and F and the right hand sides e and f as below, where e and f are the vectors $(1, 0, 0, ...0)^T$ which constrain that the empty sequence must have probability 1, and matrices E and F encode information sets.

$Ex = e$ and $Fy = f$

We can then represent the payoffs for players 1 and 2 by matrices A and B respectively. These matrices can be thought of as having on one dimension all of player 1's possible sequences (so has dimension $|S_1|$) and on the other dimension all of player 2's possible sequences (so has dimension $|S_2|$). Then each cell in the matrix represents the payoff associated with the leaf node defined by the combination of actions taken by player 1 and 2 under the pair of sequences. We note that this matrix will be very sparse as most combinations of moves corresponding to cells will not in fact be represented as a leaf in our game tree. We further note that we have so far ignored the nature player in this analysis. Given that nature is also taking some actions in our game tree, each pair of sequences may actually map to more than one cell in a payoff matrix, in which case we say the payoff is the average (using natures probabilities) across these cells.

If player 1 and player 2 are playing realization strategies x and y respectively we can characterize the expected payoff for player 1 as $x^T Ay$ and $x^t By$ where we recall that since we are in a zero-sum game these must sum to zero (or equivalently A = -1 * B). So if we consider the realization plan y to be fixed, then x is a best response to y if and only if it is an optimal solution to the linear program:

$\max_x x^T(Ay)$ subject to

$x^T E^T = e^T$, $x \geq 0$

and we can similarly define this for player 2.

In our context this can be interpreted as saying that, given a games minimax value of v', then the set of Σ_1^ϵ consisting of realization plans representing ϵ-safe strategies, can be represented as:

$\Sigma_1^\epsilon = \{x : \exists v s.t. f \cdot v \geq v' - \epsilon, F^T v \leq Ax, Ex = e, x \geq 0\}$

From the above we can see that the optimization oracle required by the below algorithm, where for a given adversarial strategy we must calculate a best response strategy, is a linearly solvable system. Furthermore we see that the expected payoff for a player, given a fixed adversary is a linear function, which is also required in the below.

5.2 Exploit

In order to select amongst our possible strategies the algorithm maintains an accumulated cost vector c (one entry in the vector per strategy) where each entry in the vector represents the accumulated observed cost of having chosen the associated strategy. This vector is built up during explore iterations as below (note that we ignore the cost associated with exploit iterations for reasons discussed below).

Given a c, in order to choose amongst our set of possible strategies on round t, we use our optimization oracle to solve for k, the equation $(c + \tau) \cdot k$, where τ is a random perturbation vector and c is the accumulated cost vector from rounds 1 to $(t-1)$ (bearing in mind that actually we would have only updated this cost vector on a round which was a explore iteration). This can be thought of letting τ be a random vector which in some sense represents our expected cost vector for this iteration (which we don't have access to until we have made our choice) - it is clear that if $\tau = c^t$ where c^t is the cost vector on the tth round, then we would have chosen the optimal k from our set of options.

One further aspect is that, in the case of poker, we don't actually get to observe c^t on round t, and instead we only get to observe $c^t \cdot k$, the reward given our choice of k. So in this case we can't simply keep a accumulated cost vector since we only see the cost associated with one specific choice from K at each iteration. The approach taken here is to use instead an estimator for c, which is $< 0, ..., \frac{n\delta_t}{\gamma}, ..., 0 >$ where $\delta_t = c^t \cdot k$ and gamma is a parameter of the algorithm and n is the dimensionality of our strategy space, which is linear in the number of game states. In other words we estimate 0 as the cost vector for all strategies not chosen, and use a normalized cost for our chosen strategy.

5.3 Explore

The common approach in an explore phase of an algorithm is to select amongst all possible strategies (possible using some distribution), and play using this strategy. This is done so that we can refine our overall understanding of how each choice performs using the observed outcome of our exploration choice.

The optimization used in this algorithm is based on the fact that the reward function over our strategies is linear. This means that if we choose a basis for our strategy space and explore only those strategies in our basis, given the cost associated with just one of our basis choices, we can use linear interpolation to estimate the cost associated with all of the strategies in our space which are by definition linear combinations of those strategies which form our basis.

In order for this approach to yield estimations for our non-basis strategies with reasonable error, it is important that we choose our basis in such a way as to ensure that when we extrapolate from the basis to each of our strategies we don't multiple the sampling error associated with the observed basis strategy cost, too much. To ensure this the algorithm requires that our basis is a 2-approximate barycentric spanner [6], which is a basis where every element of our set is a linear combination of the basis with coefficients at most 2.

It seems clear that choosing such a basis will indeed minimize the possible blow up of error when estimating the error on each of our strategies across our strategy space, based on linear interpolation of the cost associated with the basis. Furthermore it is possible to construct such a basis using the optimization oracle as defined above, using a $O(n^6 logn)$ run-time algorithm that makes repeated calls to the oracle [6]. This approach means that we can get good estimates for the cost associated with each of our strategies based on a number of samples which depends on the size of the basis, rather than the size of our whole set. In the case of realization plans, as above, the dimension of this basis is $O(|S_1|)$. We note that the reason that the observed reward is not used in the case of an exploit iteration is that it was unclear to the algorithms authors as to how to incorporate this into the unbiased estimate across all of our strategies, since in this case we would have the cost associated with a specific strategy that may not be in our basis.

5.4 Summary of Algorithm

So in summary, our algorithm performs as follows:

- We specify a set of realization plans, that correspond to the set of ϵ-safe strategies for a given ϵ. This set of possible strategy choices is a convex and compact set $K \subseteq R^n$.
- Using an optimization oracle we determine a 2-approximate barycentric basis B for K.
- At each iteration we toss a γ biased coin to determine if we explore or exploit.
- On an explore iteration the learner randomly chooses $k \in B$ and the adversary simultaneously chooses a loss vector $l \in R^n$ which specifies the loss associated with each possible choice in K. In a imperfect information setting (such as ours) the learner does not get to observe l, but instead gets to observer $k \cdot l$, which represents the loss associated with its specific choice. We use the estimator above to generate an estimate for l and use this estimate to update our accumulated cost vector, applying linear interpolation to update the cost associated with every linear combination of our chosen basis strategy.

- On an exploit iteration the learner chooses $kinK$ using a follow the perturbed leader approach, using the optimization oracle to solve for our accumulated cost vector with a random perturbation vector standing in for the unknown cost vector associated with this iteration. The adversary also chooses a reward vector l in this instance, but it is ignored.

- We repeat for T rounds.

In a full information settings (where the learner gets to observe the entire loss vector $l \in R^n$ rather than just $k \cdot l$), it has been shown that if we have access to an optimization oracle which given a loss vector $l \in R^n$, can compute the optimal $k \in K$, then the bandit linear optimization problem has an solution whose regret grows with $O(T^{\frac{2}{3}})$, and hence converges to an optimal solution.

In a imperfect information settings, where we only get to observe the cumulative loss associated with our specific choice (so we observe $l \cdot k$), with a choice of $\gamma = nT^{\frac{-1}{3}}$ and $\epsilon = \frac{(\frac{\gamma}{T})^{\frac{1}{2}}}{2M}$ where M is the diameter of the set of possible rewards, it has been shown [7] that we can construct an estimate of c^{t-1} which is sufficiently accurate as to bound our regret in a similar fashion. The overall run-time of our algorithm is then $O(n^6 logn + Tn^3.5)$ coming from the requirement to determine our 2-approximate barycentric basis before executing the algorithm.

5.5 Analysis

The authors give two theorems associated with ϵ-safe learning, which are stated as per the paper, and then interpreted below.

Theorem 1: *An ϵ-safe learner is guaranteed to, in expectation win no less than ϵ less than the minimax value on any round against any opponent. Furthermore, against any non-adaptive adversary its win-rate will approach the minimax value as the length of the match increases.*

Theorem 2: *For any $\epsilon > 0$, using a 0-safe learner weakly ϵ-dominates playing an arbitrary static equilibrium strategy in a repeated game given a sufficiently large T.*

The first of these theorems is straightforward and follows directly from the definition of our learner. Since the learner is only selecting between ϵ-safe strategies, it has the same guarantee associated with any ϵ-safe strategy. If we are playing a static or non-adaptive strategy, then given that the learner has the usual no regret property, we can see that it will converge to a best response strategy (within our restricted strategy space of ϵ-safe strategies) which will have a minimax value.

The second theorem states that if we use a 0-safe learner, which is a learner whose choices are restricted to the set of equilibrium strategies, then our choice relative to an arbitrarily chosen strategy must be weakly dominant. Whilst no formal proof is given in the paper, this seems reasonable as our learner can choose between all possible equilibrium strategies and does so in such a way as to minimize regret in terms of the optimal choice of equilibrium strategy.

The authors note that despite the strong worst case guarantees of ϵ-safe learning, in practice the algorithm did not perform well. Using $\epsilon = 5$ the learner wins at a rate of 60.14 mb/h, a small 0.5% improvement over 3,000 hands. If the guarantee is further weakened to 50, the learner improves by a further 1.52 mb/h. The authors conclude that with this approach, 3,000 hands may be insufficient time to sufficiently learn to exploit an opponent. They further conclude that different approaches that consider different estimators with lower variance, or varying the value of ϵ over the course of a match may yield better improvements, although this latter approach would be costly as it would involve generating a new barycentric basis for each new value of ϵ. It is also noted that there is an alternative to the above algorithm, the self-concordant learner, which doesn't require a 2-approximate barycentric spanner and hence could be used with this approach with out this overhead when changing the value of ϵ.

6 Equilibrium Restricted Learning

With this approach the authors considered further relaxation of the strong safety guarantees given above. As observed above in the analysis of the baseline online learner, one reason why this learner takes many hands to converge to an optimal strategy is that it has no prior over its set of possible actions at each information set, so in some sense must both learn to play poker as well as calculate an optimal strategy. The authors note that in the real world what often characterizes a good player vs. a bad player is that the good player has experience and can therefore rule out many possible actions immediately as being sub-optimal. This motivates the following approach where we use an online learner (as in the baseline above), but constrain its actions at each information set to those which are considered to be of non-trivial under some equilibrium strategy (provided as a parameter to the algorithm).

With this approach the authors hope to balance the fact that constraining the learner at each information set may mean that its set of possible overall strategies no longer includes those that can fully exploit an opponent, with the idea that by constraining using an equilibrium strategy, we can still make some weak guarantees about worst case performance. This approach will yield a learner with at least some good strategy in its space (since the equi-

librium itself is an option), and furthermore ensures that no dominated strategy (a strategy for which, regardless of an opponents strategy, there exists a strategy with a better expected payoff) can exist in the decision space. The authors also note that this will also preclude iteratively dominated strategies from the learners strategy space, which is less optimal, as although such a strategy should never be played against a rational opponent, it may be the case that our opponent is not rational.

6.1 Implementation of Algorithm

As in the baseline algorithm above, the authors use the Exp3 algorithm with the GL algorithm as its full information underlying learner. A learner is implemented at each information set, with a set of choices constrained as above by an equilibrium strategy which is a parameter to the algorithm. The algorithm also takes an ϵ parameter which defines the cut-off point for considering actions associated with the provided equilibrium strategy, and a learning rate α which controls the learning rate of the underlying GL learner.

The authors modify the Exp3 algorithm to better suit this context as follows. Instead of taking one exploration parameter γ, instead 4 parameters are provided $\gamma^{max}, \gamma^{min}, \alpha$ and β. This control the exploration parameter over time where if we are on a time step less than α we use γ^{max} as our exploration parameter, on time steps between α and β we use a linearly interpolated exploration rate (interpolated between γ_{max} and γmin) and after time step β we use rate γ_{min}. This was done to match the intuition that we may with to aggressively explore in initial rounds of play, but become more conservative over time as our learner has had more observations on which to base its weights.

The authors also modify the Exp3 algorithm to explore its set of possible actions at each information set based on their weights under the provided equilibrium strategy, rather than by using the default uniform distribution. This could be thought of as providing a prior over its possible choices which bias it towards making the same decisions as the equilibrium strategy, whilst over time as the learner accumulates more observations it can start to move away from this prior distribution towards one which may better exploit an opponent. In other words for early rounds of the game where we have a high explore parameter, we will be playing in a similar fashion to the original equilibrium strategy (since we are sampling from the set of actions available to the equilibrium strategy using the same distribution as that employed by the equilibrium strategy), whilst in later rounds we become more dependent on the learned distributions that may better exploit our opponents weaknesses.

6.2 Analysis

The authors note that one benefit of restricting the learner in this way is the following theorem.

Theorem 3: *Against an equilibrium opponent, an equilibrium restricted learner will achieve the minimax value in expectation on every round.*

This follows from the observations above that since we are sampling our strategy from an equilibrium strategy, and that an equilibrium strategy is a best response to any equilibrium opponent, our learner is guaranteed to play an optimal response against an equilibrium learner. Note that this is a weaker guarantee that that of the previous section since it only gives us a worst case in the case where our opponent is playing an equilibrium strategy (which it may well not be). Another way of thinking about this is that every choice our learner makes, when sampled from the equilibrium strategy, will have minimal regret, so it will never have reason to update its weights away from those given by the equilibrium strategies distribution. The authors further note that another advantage to this approach (over the previous ϵ-safe approach) is that it has low computational overhead, as sampling from and updating the learners strategy is equivalent computationally from sampling from any fixed strategy. This may make this approach more reasonable in larger games.

The authors found (through experimentation) that aggressive exploration parameters and learning rates performed best, and used $\gamma^{max} = 1, \gamma^{min} = 0.01, \alpha = 10$ and $\beta = 25$. Over 300 hands the equilibrium restricted learner placed first in the evaluation competition with a win-rate of 88.5 mb/h, and over 3,000 hands had a win-rate of 102.9 mb/h. From these results it seems reasonable to draw the conclusion that a learner which is restricted in this fashion, but is still able to adapt to a given adversary, can both accelerate learning and play strong and safe strategies against fixed adversaries.

Finally the authors noted that increasing match length beyond 3,000 did not yield significant further improvement. This was due to restrictions placed on those actions that could be taken by the learner, with specific optimally exploitative strategies for a given opponent, not being available to it. They conclude that different approaches that modify the learners restrictions may prove to be fruitful, for example only removing dominated strategies, or allowing strategies that may be non-trivially played under any equilibrium strategy.

7 Conclusion

The authors conclude that whilst learning adaptive strategies in these large games is hard, it can yield significant improvement. Given the size of the game states in poker, modern online

learning methods do not provide adequate solutions within a reasonable number of hands, but by combining such online learning methods with known good strategies, the authors were able to achieve exploitative strategies that still maintained some worst-case guarantees.

8 References

[1] K. Waugh, D. Schnizlein, M. Bowling, and D. Szafron. Abstaction Pathologies in Extensive Games.
[2] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret Minimization in Games with Incomplete Information.
[3] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling. Monte Carlo Sampling for Regret Minimization in Extensive Games.
[4] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a Rigged Casino: The Adversarial Multi-arm Bandit Problem.
[5] C. Allenberg-Neeman and B. Neeman. Full Information Game with Gains and Losses.
[6] B. Awerbuch and R. Kleinberg. Online Linear Optimization and Adaptive Routing.
[7] P. McCracken and M. Bowling. Safe Strategies for Agent Modeling in Games.

9 Appendix

Rules of Leduc Hold'em:

Leduc Holdem is a two player poker game. There are only six cards in the deck, two jacks, two queens and two kings (there are no suits), and is shuffled before each hand. A hand begins with each player betting one chip (the ante) and receiving one card which only they can see. There is then a round of betting (so this round of betting is based only on a players private card), and when this betting round has completed one community card is turned over (so this card is shown to both players and is shared between the players). There is then a further round of betting, followed by the showdown. At the showdown, if either player has paired their private card with the community card they win (only one player can do this), otherwise if no player has a pair, then the player with the higher card wins. If both players have the same value private card, it is a draw and the pot is split evenly amongst the players.

In the first round of betting, the first player to play (each player takes turns being the first player to play) can either bet or check. If they bet, they must add 2 chips to the pot, if they check they add no chips to the pot. The second player can then either fold, check or raise. If they fold they forfeit the game and the pot is awarded to the the first player. If

they check they must also put in two chips to match the bet of the first player. If they raise they must put in two chips to match the bet of the first player, and an additional two chips. In the latter case the first player must then play again, with the option to either check or fold. If they check they match the additional two chip bet by placing two more chips into the pot, if they fold they forfeit the game and the pot is awarded to the other player. The second round of betting is identical to the first, except that bet sizes are 4 chips instead of 2.