This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

Opponent Modelling by Expectation-Maximisation and Sequence Prediction in Simplified Poker

Richard Mealing and Jonathan L. Shapiro

Abstract

We consider the problem of learning an effective strategy online in a hidden information game against an opponent with a changing strategy. We want to model and exploit the opponent and make three proposals to do this; firstly, to infer its hidden information using an expectation-maximisation algorithm, secondly, to predict its actions using a sequence prediction method, and finally, to simulate games between our agent and our opponent model in-between games against the opponent. Our approach does not require knowledge outside the rules of the game, and does not assume that the opponent's strategy is stationary. Experiments in simplified poker games show that it increases the average payoff per game of a state-of-the-art no-regret learning algorithm.

Index Terms

Opponent modelling, expectation-maximisation algorithms, sequence prediction, counterfactual regret minimisation, simplified poker, learning in games.

I. Introduction

The question of how to make a learning agent that can play a game with hidden or incomplete information, such as poker, is an ongoing and challenging problem (see, for example, the Annual Computer Poker Competition 2014 results [1]). In a two-player game with perfect information

R. Mealing and J. L. Shapiro are with the Machine Learning and Optimisation Group in the School of Computer Science at The University of Manchester, M13 9PL, UK, E-mails: {mealingr,jls}@cs.man.ac.uk.

Manuscript received October 21st, 2014; revised March 8th, 2015, June 30th, 2015, and September 26th, 2015; accepted October 7th, 2015.

(i.e. where each player knows all prior events), such as backgammon or go, the optimal strategy is deterministic. It can often be learned with conventional methods such as backwards induction. Poker, however, requires you to be unpredictable to play optimally (e.g. by bluffing), which can only be expressed using a "mixed strategy" in the language of game theory. It has been known for a long time that learning an optimal mixed strategy is difficult [2]–[4]. Although there have been many advances to this problem, particularly for playing a learning algorithm against itself in a process called "self-play" [5]–[7], it is still challenging especially in large games with many hidden states.

Our problem is that of learning an effective strategy online in a hidden information game against an opponent with a changing strategy. Our approach is to model the opponent, and to use this model to improve our strategy. The purpose of an opponent model is to predict the opponent's actions given its information. Thus, to learn an opponent model, we must observe the opponent's actions with its corresponding information. However, in our case, it has hidden information, which may only be partially revealed at the end of each game. The actions of a typical opponent will give indications of its hidden information e.g. often betting with strong hands and folding with weak hands. Our first proposal is then to infer its hidden information, when it is not revealed, based on its actions using *expectation-maximisation*. This is an iterative procedure to compute maximum likelihood estimates of model parameters given partially observed data. In our case, the model is of the opponent's strategy, the observed data is the opponent's actions given our information (public actions and our hidden information), and the hidden data is its hidden information. We do not assume that the opponent's strategy is stationary. Our second proposal is then to use *sequence prediction* to predict a changing opponent strategy such that for each of its decision points, identified using its inferred hidden information, its actions are predicted using its actions at that point from previous games. Sequence prediction finds effective predictive contexts amongst different interaction memories. Finally, we need to decide how to use our opponent model to improve our strategy, which is more difficult if it has inaccuracies. Our third proposal is then to *simulate games* against our opponent model. If our agent learns from games, then this will improve its strategy against our opponent model, which if accurate will improve its strategy against the opponent. Simulating games is advantageous as it lets us control computational cost, control our reliance on our opponent model, and use any algorithm that uses game results.

In short, we make three proposals, which can be used online; two for building an opponent model, specifically to handle hidden information as well as changes in the opponent's strategy, and one for using an opponent model, which may have inaccuracies. Our proposals are as follows:

1) To use expectation-maximisation to infer the opponent's hidden information when it is not revealed.
2) To use sequence prediction to model the opponent's strategy and predict its actions based on its inferred hidden information and its actions from previous games.
3) To simulate games against our opponent model in-between games against the opponent to improve learning.

We use a state-of-the-art no-regret learning algorithm to update our strategy using rewards from actual and simulated games. If our opponent model is completely accurate, then playing a best-response strategy against it would maximise our expected rewards. However, it is unlikely to be completely accurate, particularly near the start with data from only a few games. This matters as Johanson et al. showed that even a slightly inaccurate best-response strategy can give very low expected rewards [8]. This proposal will exploit the opponent less if our opponent model is completely accurate, but is likely to be less exploitable if it is inaccurate. Many opponent models require knowledge outside game rules, or assume a stationary opponent strategy, or both. Our opponent model has several advantages: 1) it can be built and used online; 2) it does not require knowledge outside the game rules; 3) it can infer the opponent's hidden information via expectation-maximisation; 4) it can predict the actions of an opponent with a changing strategy via sequence prediction, and 5) it can be used with any strategy update method that only requires results from games.

We test our proposals in a pair of two-player simplified poker games against various opponents. However, our proposals can be used with more than two agents by modelling each agent separately and training against all of them. Our primary idea is that our proposals will give higher average payoffs per game than not using them. Our secondary ideas are as follows. Firstly, that inferences of the opponent's hidden information based on its behaviour using expectation-maximisation will give higher average payoffs per game in our approach than inferences ignoring its behaviour. Secondly, predictions of the opponent's actions using a sequence prediction method will give higher average payoffs per game in our approach than predictions using empirical

probabilities. Experiments in the pair of simplified poker games measuring the change in our agent's average payoff per game confirm these ideas.

II. Related Work

A large part of opponent modelling research in games with hidden information, otherwise known as *imperfect information* games, has focused on poker due to its huge popularity. Some approaches use domain-specific heuristics and expert knowledge. For example, Billings et al. propose a multi-player Texas hold'em agent named Loki whose strategy is based on poker-specific heuristics i.e. effective hand strength, which is calculated using hand strength, hand potential, pot odds, and opponent models [9]. Other approaches use large databases of human play. For example, the opponent modelling by Billings et al. [9] is improved by Davidson et al. through experiments with neural networks trained on hands played in the Internet Relay Chat (IRC) poker server [10]. A second example is by Ponsen et al. where they use games played in an online multi-player no-limit Texas hold'em room to learn a relational regression tree-function to adapt prior opponent models to specific opponents [11]. A third example is by Broeck et al. where they apply Monte-Carlo Tree Search (MCTS) to multi-player no-limit Texas hold'em and learn opponent models using games played in an online casino [12]. A final example is by Rubin and Watson, where they look at a two-player limit Texas hold'em agent named SARTRE (Similarity Assessment Reasoning for Texas hold'em via Recall of Experience), which acts by re-using solutions similar to its situation from a large database of human poker hands [13].

Many approaches use Bayesian probabilistic models. For example, Korb et al. propose a Bayesian Poker Program for two-player five-card stud poker, which learns through experience using a Bayesian network to model each player's hand, opponent behaviour conditioned on its hand, and betting curves that govern play given a probability of winning [14]. A second example is by Southey et al. where they propose a Bayesian probabilistic opponent model for two-player poker games, which infers a posterior opponent strategy given a prior and observations of its play [15]. A final example is by Baker and Cowling, where they use Bayesian opponent modelling in multi-player one-card poker to classify each opponent based on its behaviour as loose or tight, as well as passive or aggressive, and to counter the most dangerous type [16].

Another set of approaches use best-response strategies, or approximate Nash equilibrium strategies, or both. For example, Risk and Szafron use approximate Nash equilibrium strategies in

three-player limit Texas hold'em, which they find using counterfactual regret minimisation [17]. Two more examples are by Johanson et al. firstly using Restricted Nash Response (RNR) strategies, and secondly using Data Biased Response (DBR) strategies, the latter being an enhancement of the former, which they also find using counterfactual regret minimisation. RNR and DBR strategies tradeoff between exploiting an opponent and being exploitable by solving a modified game to potentially achieve strategies with lower exploitability for a given degree of exploitation [8], [18]. Ponsen et al. use Monte-Carlo sampling to speed up the convergence of RNR strategies [19]. A fourth example is by Bard et al. where they compute a set of RNR and DBR strategies against certain opponents offline and find the mixture that maximises its expected reward online using a multi-armed bandit algorithm [20]. A final example is by Ganzfried and Sandholm, where they propose Deviation Based Best-Response, which initialises prior opponent action distributions as if it has played a number of fictitious hands according to an approximate Nash equilibrium strategy, and then updates them through observations of its play. It uses these posterior distributions to compute an opponent model that is close to the approximate Nash equilibrium, making it less exploitable, and plays a best-response strategy against it [21].

For more information we refer the reader to the review by Sandholm on the state of solving incomplete-information games [22], and the review by Rubin and Watson on algorithms, approaches, and agents in computer poker [23]. Our expectation-maximisation algorithm is related to approaches that use Bayesian probabilistic models in that it makes use of Bayes' rule. Additionally, the state-of-the-art no-regret algorithm that we use is based on counterfactual regret minimisation, which is an algorithm that is also used by [8], [17]–[20] to calculate best-response strategies, approximate Nash equilibria, and combinations between both. Our work differs from [9]–[14], [16] in that we avoid using knowledge outside the rules of the game and update our opponent model online using only information accessible to our agent. Our work also differs from [8]–[10], [15]–[21] in that we do not assume that the opponent uses a stationary strategy. One advantage of these differences is that it makes our work applicable to more opponents and more imperfect information turn-based games (or situations that can be modelled as such). Another advantage is that by simulating games against the opponent model, instead of immediately playing a best-response strategy against it, which can be brittle [8], our strategy will be more robust to inaccuracies in the opponent model. Out of the prior exploitation approaches designed to model dynamic opponents in real-time, only the MCTS approach by

Broeck et al. reports effective results [12]. If their approach did not require prior knowledge in the form of training its opponent model using a large database of games, it could have served as a fair comparison to our approach.

III. Background

We consider two-player, zero-sum, imperfect information, turn-based games of finite length and with discrete actions. Before formally explaining our approach we need a representation for these games, which is described in Section III-A. Additionally, we want to empirically test our three proposals to see if training our agent against our opponent model in-between games against the opponent improves its average payoff per game. To do this we need: 1) candidate two-player, zero-sum, imperfect information, turn-based games of finite length and with discrete actions, for which we use two simplified poker games described in Section III-B, and 2) candidate opponents, for which we use a mixture of state-of-the-art and popular algorithms described in Section III-C. In Section III-D we describe, in general, expectation-maximisation as well as the online variant that we use as the first component in our opponent model. In Section III-E we describe, in general, sequence prediction as well as the specific method that we use as the second component in our opponent model. Finally, in Section III-F and in more detail in Appendix A, we describe counterfactual regret minimisation and the online variant that we use to update our agent's strategy.

A. Extensive-Form Game

An extensive-form game is a model of sequential decision-making and can represent these games effectively. It can be visualised as a game tree, with nodes as game states and edges as actions. At each non-terminal node a player acts or is "on turn", which means that it chooses the action to take at that node. The chosen action determines the edge that is followed to the next node. Each node has only one parent and so can be represented by a unique history or sequence of actions taken to reach it, $h = (a_1, a_2, \ldots, a_m)$, where each action, a_i, $1 \leq i \leq m$, is taken by one of the players. These actions include "chance" actions such as die rolls or card deals, which are taken by the "chance" (sometimes called "nature") player. Thus, h represents all of the information seen by an omniscient observer. The set of all nodes is H and the subset $Z \subseteq H$ contains terminal (leaf) nodes, which have no children.

If one or more actions are hidden from a player, such as the dealing of opponent cards in poker, then that player cannot be sure of what node it is at in the game tree. What it does know is that the node belongs to a subset of nodes, where each node in that subset is represented by an interleaved sequence of observed actions and actions that could represent the hidden information. For example, in poker a node could be $(A\heartsuit A\spadesuit, K\diamondsuit K\clubsuit, R)$ where player one was dealt aces, player two was dealt kings, and player one raised. At this point neither player has seen the other's private cards. From player two's perspective, this node could be any $(C_1C_2, K\diamondsuit K\clubsuit, R)$, where C_1 and C_2 are unique cards out of a standard fifty-two card deck other than the kings being held. From player one's perspective, this node could be any $(A\heartsuit A\spadesuit, C_3C_4, R)$, where C_3 and C_4 are unique cards out of a standard fifty-two card deck other than the aces being held.

This subset of nodes from a player's perspective is called an *information set* and is denoted by I. The set of all of player i's information sets is called an *information partition* and is denoted by $\mathcal{I}_i$. It is called a partition because each node belongs to exactly one information set and there are no empty information sets. If there is no hidden information, then each node belongs to its own information set. We denote the (possibly empty) set of edges or actions at a node h by $A(h)$, and the player who acts at that node by $P(h)$. Note that, in our games, the available actions and the player who acts at an information set are equal to the available actions and the player who acts at any node in that information set respectively i.e. $A(I) = A(h)$ and $P(I) = P(h)$ for any $h \in I$. Each player i has a strategy, which is a set of discrete probability distributions, one for each of its information sets where it acts over the actions available at that information set. We denote player i's strategy as $\sigma_i = \{f_{A(I)} : I \in \mathcal{I}_i \text{ and } P(I) = i\}$, where $f_{A(I)}$ is a probability mass function over the available actions at information set I, $A(I)$.

B. Games in our Experiments

We use a pair of two-player, zero-sum, imperfect information, turn-based poker games of finite length and with discrete actions in our experiments. We assume that the players in both games have perfect recall, meaning that they can remember the exact sequence of observable actions. In both of these poker games each player has, at most, three actions when it acts. It can fold (F) giving up the pot, or call (C) matching its opponent's current bet (if bets are equal, then this is also called a check and just passes the turn), or raise (R) matching and exceeding its

opponent's current bet by a fixed amount. If no one folds, then a showdown eventually occurs and the player with the best hand (of dice or cards) wins the pot.

1) Die-Roll Poker: The first game we use in our experiments is die-roll poker, which was introduced by Lanctot et al. [24] and uses dice instead of cards. The game is as follows:

1) Each player antes one chip into the pot.
2) Each player rolls its first private six-sided die.
3) First public betting round occurs, each raise (maximum of two in total) is two chips.
4) If no one folded, each player rolls its second private six-sided die.
5) Second public betting round occurs, each raise (maximum of two in total) is four chips.
6) If no one folded, a showdown occurs and the player with the highest dice sum wins the pot.

Die-roll poker has imperfect information due to each player's die rolls initially being hidden from its opponent. If the game ends in a fold, then each player's die rolls remain hidden. Otherwise a showdown occurs and the sum of each player's die rolls are revealed to its opponent, but each individual die roll that constituted that sum is not revealed. For example, at a showdown a player might reveal to its opponent that the sum of its die rolls is three, but its opponent cannot tell if the sum is either ⚀⚁ or ⚁⚀. Fig. 1 shows the game tree, including die-rolls and a betting round.

2) Rhode Island Hold'em: The second game we use in our experiments is Rhode Island hold'em, which was introduced by Shi and Littman [25] and uses a standard fifty-two card deck. Each player is dealt only one private card and only two public cards are dealt. The game is as follows:

1) Each player antes five chips into the pot.
2) Each player is dealt one private card from a standard fifty-two card deck.
3) First public betting round occurs, each raise (maximum of three in total) is ten chips.
4) If no one folded, the first public "flop" card is dealt.
5) Second public betting round occurs, each raise (maximum of three in total) is twenty chips.
6) If no one folded, the second public "turn" card is dealt.
7) Third public betting round occurs, each raise (maximum of three in total) is twenty chips.
8) If no one folded, a showdown occurs and the player with the best three-card hand wins the pot.

Rhode Island hold'em has imperfect information due to each player's private card initially being hidden from its opponent. If the game ends in a fold, then each player's private card remains hidden. Otherwise a showdown occurs and each player's private card is revealed to its opponent. Fig. 2 shows the game tree, including card deals and a betting round.

3) Bucketed Rhode Island Hold'em: This is an abstraction of Rhode Island hold'em, which reduces its number of information sets where players act from 2.50×10^7 for player one, and 2.46×10^7 for player two, to 2.52×10^3 each. This allows the agents we use to learn effective strategies within 1×10^5 games, which is the number of games we evaluate agents over in our experiments. Evidence for this is discussed in Section V-A. The abstraction uses percentile bucketing based on expected hand strength squared. Expected hand strength is the probability of a player's private cards combined with the public cards winning against a uniform random draw of the opponent's private cards combined with the public cards. Expected hand strength squared is simply the square of the expected hand strength. This gives more weight to initially weak hands that could become strong such as straights or flushes. Percentile bucketing divides all n-card hands evenly between a set of buckets, b_n, where in Rhode Island hold'em $n \in \{1, 2, 3\}$. For more information on percentile bucketing and expected hand strength see Johanson's MSc thesis [26, pp. 24–26].

C. Opponents in our Experiments

We use opponents based on popular and state-of-the-art algorithms in our experiments. These opponents are as follows:

- OS-MCCFR (without our model) by Lanctot et al. [27].
- PGA-APP (Policy Gradient Ascent with Approximate Policy Prediction), a state-of-the-art, Q-Learning based, reinforcement learning method by Zhang and Lesser [7].
- UCB (Upper Confidence Bounds), a popular adaptive bandit algorithm, see Auer et al. [28].
- CFRX (CFR with X iterations) by Zinkevich et al. [29], not an agent in itself but used to generate approximate Nash equilibrium strategies (only used in die-roll poker).

As UCB is designed for a single-state environment, we use an instance of it for each of the opponent's information sets where it acts. The average reward for each UCB instance is set to the average of the rewards from the games involving its associated information set. We measure the change in the average payoff per game of OS-MCCFR against these opponents when trained

against our opponent model. We label our agent OS-MCCFR with an Opponent Model (OS-MCCFR OM).

D. Expectation-Maximisation

We use an expectation-maximisation algorithm to infer the opponent's hidden information based on its actions. The expectation-maximisation (EM) algorithm, first proposed by Dempster et al. [30], can iteratively calculate maximum likelihood estimates of parameters in a statistical model dependent on latent (unobserved) variables. The EM algorithm alternates between an expectation (E) step and a maximisation (M) step. The E-step creates a function for the expectation of the log-likelihood evaluated using current parameter estimates. The M-step updates parameters by maximising the expected log-likelihood computed in the E-step. The new parameters are then used to determine the probability distribution of the latent variables in the next E-step and the algorithm iterates. The EM algorithm will always converge to a, possibly local, maximum likelihood estimate, which may be improved through multiple runs with different initialisations. We use what Liang and Klein refer to as a stepwise EM algorithm [31], first proposed by Sato and Ishii [32], generalised by Cappé and Moulines [33], and applied to poker by Butterworth [34]. The idea is to stochastically approximate the E-step by incorporating each new observation iteratively, whilst leaving the M-step unaltered. The stepwise EM algorithm is also guaranteed to converge to a, possibly local, maximum likelihood estimate if the step size, η_t, is restricted such that $\sum_{t=0}^{\infty} \eta_t = \infty$ and $\sum_{t=0}^{\infty} \eta_t^2 < \infty$, where t is the update number (i.e. $t = 0$ is initial, $t = 1$ is first update). Our step size is set to $\eta_t = \frac{1}{t}$.

E. Sequence Prediction

We use a sequence prediction method in our opponent model to observe and predict the opponent's actions. Its target is an opponent who changes its strategy over time (i.e. learns), but it will also work against an opponent with a stationary strategy. A sequence prediction method assumes that the probability of the future can, in general, depend on any subset of the past i.e. $\Pr(s_{t+1}|s_1, s_2, \ldots, s_t) = \Pr(s_{t+1}|H)$ where $H \subseteq \{s_1, s_2, \ldots, s_t\}$, each observation is from some alphabet $s_i \in \Sigma$ for $(t-k+1) \leq i \leq t$, and t is time. It usually has two main components, a short-term memory, and a long-term memory. Its short-term memory, S, stores the last k observations

acting as a size-k first-in-first-out stack i.e. $S = (s_{t-k+1}, s_{t-k+2}, \ldots, s_t)$, where k is the short-term memory size or lookback. Its long-term memory, L, stores conditional distributions acting as a map from observation sequences and observations to counts i.e. $L : \Sigma^i \times \Sigma \rightarrow \mathbb{N}^0$ for $0 \leq i \leq k$. The probability of an observation, $s \in \Sigma$, given a sequence of up to k observations, S', is $\Pr(s|S') = L(S', s) / \sum_{s' \in \Sigma} L(S', s')$. The sequences or conditioning contexts used for predictions depend on the sequence prediction method.

F. Outcome Sampling Monte-Carlo Counterfactual Regret Minimisation

The agent that we use to test our three proposals, to see if they can improve its average payoff per game, is a state-of-the-art no-regret learning agent based on *counterfactual regret minimisation* (CFR). CFR is a state-of-the-art algorithm which, in self-play, computes an approximate Nash equilibrium in two-player, zero-sum, imperfect information games. It works by minimising counterfactual regret in self-play, which Zinkevich et al. showed minimises overall regret, causing the average strategy profile to approach a Nash equilibrium strategy profile [29]. Before minimising an agent's counterfactual regret, it calculates the agent's counterfactual regret for not playing each of its actions at each of its information sets where it acts. An agent's counterfactual regret for not playing an action at an information set is the difference between its expected reward for playing that action at that information set and its expected reward for playing its strategy at that information set, weighted by the probability of reaching that information set if the probability of each of its actions leading to it is set to one. If an agent has a high counterfactual regret for an action, then in expectation it would have received a higher cumulative reward if it had played it more often and so the algorithm increases its probability of playing that action.

Using CFR to learn a strategy online is problematic. Firstly, calculating an agent's expected reward for playing an action requires the entire sub-tree under that action to be traversed, which is computationally costly. Secondly, if an agent action leads to an opponent action, then CFR needs that opponent's strategy to calculate the agent's expected reward for that action. Also, if an opponent action leads to an agent information set, then that opponent's strategy is needed to calculate the probability that the agent reached that information set if it had tried to do so. To tackle the first problem, Lanctot et al. proposed *Monte-Carlo* Counterfactual Regret Minimisation (MCCFR) [27], a family of sample-based CFR algorithms. MCCFR works by replacing an exact calculation of expected reward with an unbiased estimate. CFR calculates an agent's expected

reward for playing an action as the sum over each reward it could receive after playing that action multiplied by the probability of reaching the point where that reward is received. An MCCFR algorithm performs the same calculation, but only for an unbiased sample of the possible rewards. Thus, an agent's expected reward for an action is estimated by traversing only part of the sub-tree under that action, which reduces the computational cost. In expectation, MCCFR algorithms perform the same regret updates as the CFR algorithm but require more iterations. However, the cost per iteration is much lower. Generally, this speeds up convergence and makes the algorithm applicable to larger games [27]. To solve the second problem, Lanctot et al. proposed *outcome sampling* MCCFR (OS-MCCFR) [27], which is a particular sample-based algorithm that only takes one sample per iteration corresponding to the reward at the end of a game. If the rewards are sampled from games against the opponent, and it is assumed that the opponent is acting according to its true strategy, then it does not need to know the opponent's strategy and can be used to minimise regret online. We use OS-MCCFR to update our agent's strategy, and test to see if its average payoff per game is improved by simulating games between it and our opponent model in-between games against the opponent. For derivations of the key equations of OS-MCCFR see Appendix A.

IV. Our Approach

Our approach, which incorporates our three proposals, aims to improve our agent's average payoff per game by training it against our opponent model in-between games against the opponent. The first question is: how do we build our opponent model? To model the opponent's strategy we must model its hidden information. To do this, for each opponent information set I, we create a categorical distribution over its actions, a_{opp}, i.e. $\Pr(a_{\text{opp}}|I)$. The opponent information set, I, represents its knowledge, which consists of prior actions, including its private actions (hidden information), $\mathcal{H}_{\text{opp}}$, and public actions, S, $I = \{\mathcal{H}_{\text{opp}}, S\}$. At the end of a game, if we do not see the opponent's hidden information, then we do not know which information sets it acted at. In this case, for each opponent action, we consider all possible opponent information sets it could have originated from and treat it as a sample from a mixture of the associated categorical distributions. We then use an expectation-maximisation algorithm to: 1) infer a distribution over the opponent's hidden information using the categorical distributions parameters (E-step), and 2) update the categorical distributions parameters by maximising their likelihood

given the opponent's hidden information distribution (M-step). Finally, we sample the opponent's hidden information.

We model the opponent's, possibly changing, strategy by using, for each opponent information set, an instance of a sequence prediction method. Each of these predicts a distribution over the opponent's actions a_{opp} conditioned on that information set (its knowledge) I as well as a sequence of previous actions taken at that information set from previous games $(a^1_{\text{opp}}, a^2_{\text{opp}}, \dots)$ i.e. $\Pr(a_{\text{opp}}|I, (a^1_{\text{opp}}, a^2_{\text{opp}}, \dots))$. After the opponent's hidden information is either revealed or predicted, then the information sets it acted at are identified and the corresponding sequence prediction method instances observe the opponent's actions taken in them. This is explained in more detail in sections IV-A and IV-B.

The second question is: how do we use our opponent model? We could play a best-response strategy against it, but if it is inaccurate, then Johanson et al. showed that this could yield much lower rewards than expected [8]. Thus, instead our approach simulates games between our agent and our opponent model in-between games against the opponent. Each simulated game uses the sequence prediction method instances to predict the opponent's actions. OS-MCCFR updates our strategy using rewards from the actual game and simulated games. In expectation it minimises our agent's overall regret [27] against the opponent and the opponent model and gradually moves its average strategy towards a best-response strategy against them. If the opponent improves its strategy (i.e. learns towards a best-response strategy), then in expectation OS-MCCFR will reduce the exploitability of our agent's average strategy and move it towards a Nash equilibrium strategy, which has zero exploitability. Whereas there are no guarantees on the exploitability of our agent's average strategy if it always plays a best-response strategy to its opponent model. The overall process of building and using our opponent model is explained in more detail in Section IV-C.

A. Expectation-Maximisation in our Opponent Model

We want to model the opponent's strategy, σ_{opp}, which is a set of discrete probability distributions, one for each of its information sets where it acts, $\sigma_{\text{opp}} = \{f_{A(I)} : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\}$, where $f_{A(I)}$ is a probability mass function over $A(I)$. To do this we create a set of sequence predictors, $\mathcal{E}$, one for each of the opponent's information sets where it acts, $\mathcal{E} = \{p_I : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\}$. Each sequence predictor, p_I, observes the opponent's actions in

its associated information set, I, and predicts a discrete probability distribution over its future actions. The problem is that in order to know which opponent information sets the opponent acted in, we need to know its hidden information, which is only sometimes revealed at the end of a game. Thus, we wait until the end of a game before updating our opponent model. If the opponent's hidden information is not revealed, then we infer it. The observation or inference of the opponent's hidden information allows its information sets that it acted in to be identified, and the associated sequence predictors to observe the actions taken in them.

The first question is: how do we infer the opponent's hidden information? We infer the opponent's hidden information by sampling from a probability distribution over its possible instances of hidden information. Recall from Section III-A that a node or history can be represented as a unique sequence of actions taken to reach it, $h = (a_1, a_2, \ldots, a_m)$, where each action, a_i, $1 \leq i \leq m$, is taken by one of the players. An information set can also be represented as a sequence of actions, except some of those actions are hidden. For example, in die-roll poker a node could be $h = (\boxdot, \boxdot\!\!\boxdot, r, c, \boxdot\!\!\boxdot, \boxdot, f)$, where player one rolled two, player two rolled four, player one raised, player two called, player one rolled five, player two rolled three, and player one folded. At this point, neither player has seen the other's die-rolls. From player two's perspective, its information set would be $(D_1, \boxdot\!\!\boxdot, r, c, D_3, \boxdot, f) \in \mathcal{I}_2$, where D_1 and D_3 are player one's hidden six-sided die-rolls. From player one's perspective, its information set would be $(\boxdot, D_2, r, c, \boxdot\!\!\boxdot, D_4, f) \in \mathcal{I}_1$, where D_2 and D_4 are player two's hidden six-sided die-rolls.

Let player i's information set $I = \{\mathcal{H}_i, S\} \in \mathcal{I}_i$, where $\mathcal{H}_i$ is its private or hidden information and S is the sequence of actions visible to both players. Using the last example, we can write $(D_1, \boxdot\!\!\boxdot, r, c, D_3, \boxdot, f) = \{\mathcal{H}_2, S\} = \{(\boxdot\!\!\boxdot, \boxdot), (r, c, f)\} \in \mathcal{I}_2$ and $(\boxdot, D_2, r, c, \boxdot\!\!\boxdot, D_4, f) = \{\mathcal{H}_1, S\} = \{(\boxdot, \boxdot\!\!\boxdot), (r, c, f)\} \in \mathcal{I}_1$.

Using this notation, we observe our own information set $\{\mathcal{H}_{\text{pla}}, S\} \in \mathcal{I}_{\text{pla}}$, and want to infer the opponent's information set $\{\mathcal{H}_{\text{opp}}, S\} \in \mathcal{I}_{\text{opp}}$. Since we already know the public actions S, we just want to infer the opponent's hidden information $\mathcal{H}_{\text{opp}}$. Using Bayes' rule, we can infer the probability of the opponent's hidden information given our hidden information and the public actions as

$$\Pr(\mathcal{H}_{\text{opp}} | \mathcal{H}_{\text{pla}}, S) = \frac{\Pr(S | \mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \Pr(S | \mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})}. \quad (1)$$

The second question is: how do we infer $\Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})$? The probability of the public actions given the hidden information is the product of the probability of each public action given the prior public actions and the hidden information i.e.

$$\Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) = \prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}). \tag{2}$$

We can substitute Equation 2 into Equation 1 giving

$$\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S) = \frac{\Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})} =$$
$$\frac{\prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})}. \tag{3}$$

We can simplify Equation 3 by cancelling out our action probabilities as these are the same for each possible instance of the opponent's hidden information and so

$$\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S) =$$
$$\frac{\prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{opp}})^{b_i} \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}'_{\text{opp}})^{b_i} \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})},$$
$$\text{where } b_i = \begin{cases} 1 & \text{if } a_i \text{ is an opponent action} \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

The third question is: how do we calculate $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})$? The probability of the player and the opponent having particular instances of hidden information depends on the game. In die-roll poker, each six-sided die-roll is independent, thus $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) = \Pr(\mathcal{H}_{\text{pla}}) \Pr(\mathcal{H}_{\text{opp}}) = \frac{1}{6^{|\mathcal{H}_{\text{pla}}|+|\mathcal{H}_{\text{opp}}|}}$. In Rhode Island hold'em, each card draw is not independent as card draws are from the same fifty-two card deck, thus $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) = \frac{1}{|D|(|D|-1)}$, where $|D|$ is the size of the deck. For die-roll poker and Rhode Island hold'em, the joint probability of the players' hidden information is independent of what that hidden information is, meaning that $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})$ would factor out in the denominator and cancel with $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})$ in the numerator of Equation 4. However, in general this is not the case. For example, in bucketed Rhode Island hold'em, if the public cards have a high squared expected hand strength, then the probability of each player's hand being in a high bucket sequence is higher, and if a player's hand is in a particular bucket sequence, then it is slightly less likely that the opponent's hand is in the same bucket sequence. For bucketed Rhode Island hold'em, in Equation 4 we first substitute $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) = \Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}) \Pr(\mathcal{H}_{\text{pla}})$ and

$\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}}) = \Pr(\mathcal{H}'_{\text{opp}}|\mathcal{H}_{\text{pla}}) \Pr(\mathcal{H}_{\text{pla}})$ and since our hidden information is fixed then $\Pr(\mathcal{H}_{\text{pla}})$ can be factored out of the denominator and cancelled with the same term in the numerator. We calculate $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}})$ exactly by considering each card the opponent could have, counting how many times its hand is in the bucket sequence, $\mathcal{H}_{\text{opp}}$, and dividing by the number of times its hand is in any bucket sequence.

The fourth question is: how do we infer $\Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{opp}})$, where a_i is an opponent action? If a_i is an opponent action, then the opponent's hidden information, $\mathcal{H}_{\text{opp}}$, and all previous public actions, $(a_1, a_2, \ldots, a_{i-1})$, represent an opponent information set where the opponent acts, $I = \{\mathcal{H}_{\text{opp}}, (a_1, a_2, \ldots, a_{i-1})\} \in \mathcal{I}_{\text{opp}}$, where $P(I) = \text{opp}$. We could use the sequence predictor p_I to predict $\Pr(a_i|I)$. The problem with this is that it can create a sort of negative feedback loop. If the sequence predictor is inaccurate, which it probably will be initially, then its prediction of $\Pr(a_i|I)$ will be inaccurate, making the inference of $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S)$ inaccurate, which will result in the wrong sequence predictors being updated, possibly making the next prediction of $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S)$ even more inaccurate. Instead of using sequence predictors, which cannot be partially updated by making fractional observations to account for uncertainty, we use empirical probabilities, which can be.

Specifically, the EM component assumes that each distribution, $f_{A(I)}$, in the opponent's strategy, $\sigma_{\text{opp}} = \{f_{A(I)} : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\}$, is a fixed categorical distribution. The parameters of $f_{A(I)}$ are the opponent's action probabilities at I. We want to set each parameter of each $f_{A(I)}$ to its maximum likelihood estimate given our observations. If we could observe samples from $f_{A(I)}$, then maximising its parameters would be easy but, we may not know how many times each action has been played in I because if we do not observe the opponent's hidden information, then we do not know which of its information sets it acted in. Thus, instead of observing samples from $f_{A(I)}$, they are from a mixture of categorical distributions, which include $f_{A(I)}$. The maximum likelihood estimate for the probability of sampling c from a categorical distribution d given N samples from a mixture of K categorical distributions (including d) each with D categories is

$$\mu_{dc} = \frac{\sum_{n=1}^{N} \gamma(z_{nd}) x_{nc}}{\sum_{i=1}^{D} \sum_{n=1}^{N} \gamma(z_{nd}) x_{ni}}, \gamma(z_{nd}) = \frac{\pi_d \Pr(\vec{x}_n|\vec{\mu}_d)}{\sum_{j=1}^{K} \pi_j \Pr(\vec{x}_n|\vec{\mu}_j)}. \quad (5)$$

Here μ_{dc} is the probability of category c from categorical distribution d, z_{nd} is the d-th component of the 1-of-K encoded vector $\vec{z}_n$, $\gamma(z_{nd})$ is the responsibility of d to sample n, x_{nc} is the c-th component of the 1-of-D encoded vector $\vec{x}_n$, and π_d is the probability of sampling from

d. This is derived in Appendix D. We can use Equation 5 to set the parameters of the EM component's categorical distributions to their maximum likelihood estimates. Here, π_d is the probability of having played into the opponent information set associated with d, and $\Pr(\vec{x}_n|\vec{\mu}_d)$ is the probability of the opponent's action sampled from d. Thus, $\gamma(z_{nd})$ is equal to Equation 4. We can update μ_{dc} iteratively by rewriting Equation 5 as

$$\mu_{dc} = \frac{\left(\sum_{n=1}^{N-1} \gamma(z_{nd}) x_{nc}\right) + \gamma(z_{Nd}) x_{Nc}}{\left(\sum_{i=1}^{D} \sum_{n=1}^{N-1} \gamma(z_{nd}) x_{ni}\right) + \sum_{i=1}^{D} \gamma(z_{Nd}) x_{Ni}}. \tag{6}$$

We use a map from opponent information sets to real numbers $M_v : \mathcal{I}_{\text{opp}} \to \mathbb{R}$ to store the numerator of Equation 5. For example, given an opponent information set, $I \in \mathcal{I}_{\text{opp}}$, where the opponent acts, $P(I) = \text{opp}$, the probability of sampling action $c \in A(I)$ from its categorical distribution $d = f_{A(I)}$ is

$$\mu_{dc} = \frac{\sum_{n=1}^{N} \gamma(z_{nd}) x_{nc}}{\sum_{i=1}^{D} \sum_{n=1}^{N} \gamma(z_{nd}) x_{ni}} = \frac{M_v((I, c))}{M_v(I)}. \tag{7}$$

We call M_v the expected visit counts as the numerator of Equation 7 can be seen as the expected times action c is sampled from distribution d at opponent information set I, which in our case is the expected times opponent information set (I, c) is visited. Likewise, the denominator of Equation 7 can be seen as the expected times any action is sampled from d at I, which in our case is the expected times I is visited.

At the end of a game, let the opponent's terminal information set be $\{\mathcal{H}_{\text{opp}}, S\} \in \mathcal{I}_{\text{opp}}$. For each $I \in \mathcal{I}_{\text{opp}}$, that it could have acted at, $P(I) = \text{opp}$, where $I = \{\mathcal{H}_{\text{opp}}, (a_1, a_2, \ldots, a_i)\}$ and $i < |S|$, update the parameters of the categorical distribution $d = f_{A(I)}$ associated with I using the action that the opponent could have sampled from it a_{i+1} as follows:

1) E-step: Calculate $\gamma(z_{nd})$ via Equation 4.
2) M-step: Update the parameters of d via Equation 6.

For each possible path, the E-step calculates the product of the opponent's action probabilities along it (via the categorical distributions) multiplied by the probability of the opponent being dealt the hidden information along it given the player's hidden information and then normalises these probabilities, the M-step increments the visit count of each opponent information set along it by its normalised path probability from the E-step. For example, if $I = (\boxdot, D_2, r, f)$, then the E-step would calculate $\Pr(D_2|\mathcal{H}_{\text{pla}} = \boxdot, S = (r, f)) = \Pr(f|\mathcal{H}_{\text{opp}} = D_2, S =$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

$(r))/\sum_{D_2'=\boxdot}^{\boxplus} \Pr(f|\mathcal{H}_{\text{opp}} = D_2', S = (r))$. The M-step would increment $M_v((D_1, D_2, r, f))$, $M_v((D_1, D_2, r))$, ..., $M_v(())$ by $\Pr(f|\mathcal{H}_{\text{opp}} = D_2, S = (r))$. We can now sample the opponent's hidden information, h_{opp}, from $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S)$ and update the relevant sequence predictors.

$$\text{Observe action } a_{i+1} \text{ with the sequence predictor}$$
$$M_{\text{pred}}(\{h_{\text{opp}}, (a_1, a_2, \ldots, a_i)\}) \text{ for all } 0 \leq i \leq |S|,$$
$$\text{where } P(\{h_{\text{opp}}, (a_1, a_2, \ldots, a_i)\}) = \text{opp}. \tag{8}$$

M_{pred} maps from opponent information sets where it acts to sequence predictors, $M_{\text{pred}} : \{I : I \in \mathcal{I}_{\text{opp}}, P(I) = \text{opp}\} \rightarrow \mathcal{E}$.

B. Sequence Prediction in our Opponent Model

We use a sequence prediction method named Entropy Learned Pruned Hypothesis space (ELPH) by Jensen et al. [35], [36], to predict probability distributions over the opponent's future actions. Its main advantage is that it can rapidly learn a non-stationary opponent strategy, which has allowed it to be used to defeat human and agent players in simple games and will allow it to be helpful against dynamic opponents. It works by forming distributions conditioned on interaction histories of different lengths, pruning those with high entropies, and predicting using one with the minimum entropy. Given an observation, $s \in \Sigma$, it generates the set of all subsequences of its short-term memory, $\mathcal{P}(S)$, and for each subsequence creates or updates a distribution conditioned on it by incrementing the count for the subsequence and the observation in its long-term memory, $L(S', s) \leftarrow L(S', s) + 1$ for all $S' \in \mathcal{P}(S)$. It then prunes each conditional distribution (by removing its counts) if its normalised Shannon entropy, $\overline{H}$, is above a passed in threshold, H_l, for each $S' \in \mathcal{P}(S)$ $L \setminus (S', s)$ for all $s \in \Sigma$ if $\overline{H}(L(S')) > H_l$. Finally it adds the observation to the end of its short-term memory and removes the first observation if S is above its size-k limit. To make a prediction, it again gets the set of all subsequences of its short-term memory, $\mathcal{P}(S)$, and predicts using the distribution conditioned on one of these subsequences with the minimum reliable Shannon entropy, H_{rel}, $\arg\min_{S' \in \mathcal{P}(S)} H_{\text{rel}}(L(S'))$.

Our opponent model creates a set of ELPH instances, $\mathcal{E}$, one for each opponent information set where it acts, $\mathcal{E} = \{p_I : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\}$. At the end of each game, the opponent's hidden information is sampled from a probability distribution inferred using online expectation-maximisation. Using this, the opponent's information sets that they acted at during the game

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

are inferred, and the sequence predictors for them observe the opponent's actions taken in them. For each opponent information set, its associated ELPH instance observes opponent actions in it across different games and models the opponent's action distribution at it. If a dynamic opponent changes this action distribution, then the ELPH instance will rapidly learn the new distribution from its set of observation-based hypothetical conditional distributions favouring those with low entropy and high predictability.

C. Our Algorithm

Fig. 3 shows our overall algorithm and Fig. 4 shows our opponent modelling algorithm. We update our agent's strategy via OS-MCCFR [37, pp. 50] (see Section III-F & Appendix A) with rewards from games vs the real/model opponent. The time complexity of one iteration of our algorithm is dominated by the following (from most costly): 1) Simulating games. In each simulated game, at each non-terminal node, an action is sampled from a distribution, where a sequence predictor predicts each opponent distribution, and our agent updates its strategy using OS-MCCFR. This scales like $O(g[2^k d_{\text{max,\{opp\}}} + d_{\text{max,\{pla,cha\}}}]a_{\text{max}})$ where g is simulated games, k is the lookback, $d_{\text{max,N}}$ is maximum decisions in a game for players in N, and a_{max} is maximum actions at a node. 2) Sequence prediction. In general, a sequence predictor predicts using a number of distributions exponential in its lookback, which is the worst case for an ELPH instance. With a sequence predictor at each opponent information set where it acts predicting its distribution, this quickly becomes the bottleneck if the lookback grows faster than logarithmically with the game size. As shown above, this scales like $O(g2^k d_{\text{max,opp}} a_{\text{max}})$. 3) EM algorithm. After each game against the opponent it predicts probabilities and updates counts for each possible path. This scales like $O(d_{\text{max, \{opp,pla,cha\}}}|\mathcal{H}_{\text{opp}}|)$ where $|\mathcal{H}_{\text{opp}}|$ is the number of opponent hidden information possibilities. 4) OS-MCCFR. After each game it updates regrets and probabilities at each of our agent's information sets where it acted. This scales like $O(g d_{\text{max,\{pla\}}} a_{\text{max}})$.

The space complexity of our algorithm is as follows. It stores regrets and probabilities for our agent's actions at its information sets where it acts, a sequence predictor for each opponent information set where they act, which has a number of distributions exponential in its lookback, and a count for each opponent information set. This scales like $O([|\mathcal{I}'_{\text{pla}}| + 2^k|\mathcal{I}'_{\text{opp}}|]a_{\text{max}} + |\mathcal{I}_{\text{opp}}|)$ where $\mathcal{I}'_{\text{i}} = \{I : I \in \mathcal{I}_{\text{i}}, P(I) = \text{i}\}$.

Our algorithm's efficiency mainly depends on game size. Larger games have more nodes, actions, and probably information sets, requiring more space, and more observations for EM and sequence prediction to learn to given overall accuracies. Exactly how time to converge/reach given overall accuracies or required lookback scales to larger games are open questions. Also, although overall regret after a number of OS-MCCFR iterations is bounded by theory [27], we cannot yet say how our algorithm affects this. To prevent bottlenecks, a large game may need an abstraction to reduce its size, and simulated games and lookback should be set sufficiently small.

V. Results

Our experiments test if our opponent model improves the average payoff per game of OS-MCCFR against several opponents in die-roll poker and Rhode Island hold'em. We test four variations of our opponent model: 1) without expectation-maximisation or sequence prediction (UN); 2) with just expectation-maximisation (EM); 3) with just sequence prediction (SP), and 4) with expectation-maximisation and sequence prediction (EM + SP); To infer the opponent's hidden information, (EM) and (EM + SP) use expectation-maximisation (see Section IV-A), whereas (UN) and (SP) sample from $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}})$. For example, (UN) and (SP) sample die face(s) after a fold in die-roll poker with probability $1/6^{\text{round number}}$, and one card after a fold in Rhode Island hold'em with probability $1/(52 - \text{round number})$. To predict the opponent's actions, (SP) and (EM + SP) use sequence prediction (see Section IV-B), whereas (UN) and (EM) use empirical probabilities. The empirical probability of an action at an information set is the number of times it was played there divided by the total number of actions played there. Table I shows all parameters in our experiments.

A. Benefit of bucketed Rhode Island Hold'em

In Rhode Island hold'em players one and two have 2.50×10^7 and 2.46×10^7 information sets where they act respectively. This is too many for any agent we use to learn a high-reward strategy within 1×10^5 games, which is the number of games we evaluate agents over in our experiments. This is because even if an agent updates its strategy at the maximum of 6 information sets per game (3 betting rounds $\times$ 2 decisions per betting round and player), then it would take at least 4.2×10^6 games to update each information set once. Thus, making it impossible for an

agent to learn a perfect strategy in Rhode Island hold'em within 1×10^5 games. Even learning an imperfect, but effective strategy, would probably require each information set to be visited many times. Learning an effective strategy within this number of games requires learning to be generalised across information sets using an abstraction.

To test the benefit of the abstraction, we compared the average payoff per game of OS-MCCFR, PGA-APP, and UCB against a simple strategy that always raises. The abstraction reduces the number of information sets where each agent acts to 2.52×10^3 using percentile bucketing based on expected hand strength squared with five buckets for the pre-flop, flop, and turn stages in the game i.e. $b_1 = 5$, $b_2 = 5$ and $b_3 = 5$ (see Section III-B3). We found that each agent's average payoff per game is negative in the unabstracted version, and positive in the abstracted version. Thus, the abstraction allows each agent to learn to win against always raise. Linear least squares regression on the last 5×10^4 games in the unabstracted version estimates that it would take these agents 4.73×10^5, 1.22×10^6, and 1.69×10^6 games respectively to break even with zero average payoff per game. Thus, the abstraction allows these agents to learn effective strategies within 1×10^5 games. From this point for Rhode Island hold'em agents use the bucketed version and are restricted to playing strategies within it. Better strategies likely exist in larger (finer) abstractions, but would take longer to learn. An agent might perform better with a smaller abstraction as it allows them to adapt faster.

B. Performance in Die-Roll Poker and Rhode Island Hold'em

Fig. 5 shows the change in the average payoff per game of OS-MCCFR with the four variations of our opponent model. Firstly, it is always better with (EM) rather than with (UN) except in die-roll poker against CFR0. This is because CFR0 plays actions uniformly at random and so its strategy does not depend on its hidden information. This supports our first secondary idea, showing that inferences of the opponent's hidden information based on its behaviour using expectation-maximisation give higher average payoffs per game than inferences ignoring its behaviour. Secondly, it is always better with (SP) rather than with (UN) or with (EM). This supports our second secondary idea, showing that predictions of the opponent's actions using sequence prediction give higher average payoffs per game than predictions using empirical probabilities. Finally, it is always increased with (EM + SP), supporting our main idea, showing

that playing extra games between our agent and our opponent model improves our agent's average payoff per game.

Using our results, we want to estimate how OS-MCCFR with each of the four variations of our opponent model will perform in the long term. To estimate long-term average payoffs per game, we fitted exponential functions of the form $f(x) = ae^{-bx} + c$ to model the average payoffs per game. Here $f(x)$ is the average payoff per game, x is the game number divided by 1×10^5 (the number of games), and a, b, and c are parameters. We are particularly interested in the c parameter, which represents the asymptotic average payoff per game, as well as the number of iterations it takes to get close to c. We fitted these functions using MATLAB's Trust-Region-Reflective Least Squares algorithm with Bisquare weights, which is a non-linear least squares regression method found in its Curve Fitting Toolbox [38]. Table II shows each estimated c parameter and the estimated iterations to reach 99% of c. The c estimates reflect our results, showing that (EM) is always better than (UN) (except against CFR0), (SP) is always better than (UN) or (EM), and (EM + SP) always increases average payoffs per game. This implies that our approach will continue to improve average payoffs per game in the long-term.

The average payoff per game of (EM + SP) is not statistically significantly greater than that of (SP) in Rhode Island hold'em against PGA-APP and UCB. This could be because it takes longer to learn in Rhode Island hold'em as, firstly, even abstracted it has more information sets, and secondly, it has more hidden information ($5^3 = 125$ bucket sequences vs $6^2 = 36$ die rolls), which causes noisier play. In general, the EM component accuracy depends on the accuracy of its categorical distributions (one per opponent information set where it acts), so with more opponent information sets where it acts (due to more actions or hidden information) the more categorical distributions there will be, increasing learning time. To test this, we measured the difference in the average payoff per game between (EM + SP) and (SP) against OS-MCCFR, PGA-APP and UCB in die-roll poker with an increasing amount of hidden information (die faces). Table III shows that as we increase die faces, the difference decreases. Also, expectation-maximisation offers no advantage in Rhode Island hold'em if it infers the opponent has the same bucket as the agent as this indicates they have the same chance of winning.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

VI. Conclusions and Future Work

We propose an online opponent modelling algorithm that needs no knowledge outside game rules, and does not assume a stationary opponent strategy. Building it has two proposals: an expectation-maximisation algorithm to infer the opponent's hidden information in an imperfect information game, and a sequence prediction method to specialise in predicting an opponent's changing strategy. Using it has a third proposal: simulating games between our agent and our opponent model in-between games against the opponent. Experiments in simplified poker games show that our approach improves the average payoff per game of a state-of-the-art no-regret learning agent based on counterfactual regret minimisation. They indicate that our approach would improve performance in similar situations where opponents are exploitable, hidden information possibilities are sufficiently small, and iterations are sufficiently large. Future work will look at optimising the expectation-maximisation, increasing training with model accuracy, and larger domains, e.g. Texas hold'em, which may require scalability improvements and further abstractions.

Appendix

A. Counterfactual Regret Minimisation

The Counterfactual Regret Minimisation (CFR) algorithm proposed by Zinkevich et al. [29] is a state-of-the-art no-regret algorithm for two-player, zero-sum, imperfect information games which, in self-play, minimises the maximum counterfactual regret over all information sets and actions. By minimising counterfactual regret, they proved that it minimises overall regret and converges towards a Nash equilibrium.

1) Counterfactual Value: Player i's counterfactual value of information set $I \in \mathcal{I}_i$ given strategy profile σ is

$$v_i(I|\sigma) = \sum_{h \in I} \Pr(h|\sigma_{-i}) \overline{u}_i(h) \tag{A.9}$$

where $\overline{u}_i(h) = \sum_{z \in Z[h]} \Pr(z[h]|\sigma) u_i(z)$, $\Pr(z[h]|\sigma)$ is the probability of reaching node z from node h given strategy profile σ, $\Pr(h|\sigma_{-i})$ is the probability of reaching node h given strategy profile σ except player i's action probabilities are all set to one, and $Z[h]$ is the set of terminal nodes reachable from h.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

2) Counterfactual Regret: Player i's counterfactual regret for not playing action $a \in A(I)$ at information set $I \in \mathcal{I}_i$ is

$$r_i(I,a) = v_i(I|\sigma_{I\to a}) - v_i(I|\sigma), \tag{A.10}$$

where $\sigma_{I\to a}$ is the same as σ except action a is always played at information set I. With positive regret player i prefers action a rather than its strategy, with zero regret it is indifferent, and with negative regret it prefers its strategy.

3) Regret Matching: Regret matching is used to update each action probability at each information set as follows

$$\sigma_i^{T+1}(I,a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a'\in A(I)} R_i^{T,+}(I,a')} & \text{if denominator } > 0 \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases}, \tag{A.11}$$

where $R_i^{T,+}(I,a) = \max\left(\sum_{t=1}^{T} r_i^t(I,a), 0\right)$, $\sigma_i^{T+1}(I,a)$ is player i's probability of playing action a at information set I at iteration $T+1$, $r_i^t(I,a)$ is player i's counterfactual regret of not playing action a at information set I at iteration t, $R_i^{T,+}(I,a)$ is the maximum of zero and player i's cumulative counterfactual regret of not playing action a at information set I between times $t=1$ and $t=T$. For the CFR algorithm, one iteration calculates the counterfactual regrets for all of player i's actions at all of its information sets, updates its cumulative counterfactual regrets, and uses them with regret matching to update action probabilities.

B. Monte-Carlo Counterfactual Regret Minimisation

The Monte-Carlo Counterfactual Regret Minimisation (MCCFR) family of algorithms proposed by Lanctot et al. [27] are each the same as the CFR algorithm except they replace exact expected rewards with unbiased estimates. The number of iterations required for convergence increases but each iteration is faster and so convergence time generally decreases [27].

Sampled Counterfactual Value: Player i's *sampled* counterfactual value of information set $I \in \mathcal{I}_i$ given strategy profile σ is

$$\tilde{v}_i(I|\sigma, Q_j) = \sum_{h\in I} \Pr(h|\sigma_{-i})\tilde{u}_i(h|Q_j) \tag{A.12}$$

where $\tilde{u}_i(h|Q_j) = \sum_{z\in Q_j\cap Z[h]} \frac{1}{q(z)} \Pr(z[h]|\sigma)u_i(z)$, Q_j is a subset of terminal nodes $Q_j \subseteq Z$ sampled by MCCFR with probability $q_j > 0$ from $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_j, \ldots, Q_{|\mathcal{Q}|}\}$, $\bigcup_{Q_j\in\mathcal{Q}} Q_j = Z$ and $q(z) = \sum_{j:z\in Q_j} q_j$ is the probability of sampling terminal node z.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

C. Outcome Sampling Monte-Carlo Counterfactual Regret Minimisation

The Outcome Sampling Monte-Carlo Counterfactual Regret Minimisation (OS-MCCFR) algorithm defines the set of subsets of terminal nodes, $\mathcal{Q}$, such that each subset contains exactly one terminal node, i.e. $|Q_j| = 1$ for all $Q_j \in \mathcal{Q}$. This means that, on each iteration, only one terminal node is sampled, and the information sets along the path from the root to it are updated. The probability of sampling a terminal node, $q(z)$, is then equal to the probability of sampling the subset that contains that terminal node, $q(z) = q_j$. The probability distribution, or sampling scheme, is selected such that $q(z) = q_j = \Pr(z|\sigma')$. The sampled counterfactual value is then calculated as $\tilde{v}_i(I|\sigma, Q_j) = \sum_{h \in I} \Pr(h|\sigma_{-i})\tilde{u}_i(h|Q_j)$

$$
\begin{aligned}
&= \sum_{h \in I} \Pr(h|\sigma_{-i}) \left(\sum_{z \in Q_j \cap Z[h]} \frac{1}{q(z)} \Pr(z[h]|\sigma) u_i(z) \right) \\
&= \frac{\Pr(h|\sigma_{-i}) \Pr(z[h]|\sigma) u_i(z)}{q(z)} = \frac{\Pr(h|\sigma_{-i}) \Pr(z[h]|\sigma) u_i(z)}{\Pr(z|\sigma')} \\
&= \frac{\Pr(h|\sigma_{-i}) \Pr(z[h]|\sigma_i) \Pr(z[h]|\sigma_{-i}) u_i(z)}{\Pr(z|\sigma'_i) \Pr(z|\sigma'_{-i})} \\
&= \frac{\Pr(z[h]|\sigma_i) \Pr(z|\sigma_{-i}) u_i(z)}{\Pr(z|\sigma'_i) \Pr(z|\sigma'_{-i})} \approx \frac{\Pr(z[h]|\sigma_i) u_i(z)}{\Pr(z|\sigma'_i)}.
\end{aligned} \tag{A.13}
$$

Since Q_j only contains one terminal node (i.e. $|Q_j| = 1$), and the probability of reaching this terminal node $\Pr(z \in Q_j|\sigma)$ is zero for all nodes in I except one, the sums can be dropped. The probability of reaching a node given a strategy profile, can be factored into the probability of reaching that node given player i's strategy multiplied by the probability of reaching that node given the other players' strategies i.e. $\Pr(z[h]|\sigma) = \Pr(z[h]|\sigma_i) \Pr(z[h]|\sigma_{-i})$ and $\Pr(z|\sigma') = \Pr(z|\sigma'_i) \Pr(z|\sigma'_{-i})$. Finally, by assuming that the sampling strategy profile for the other players is approximately equal to their actual strategy profile i.e. $\sigma'_{-i} \approx \sigma_{-i}$ we arrive at the final equation. This equation for the sampled counterfactual value only depends on the player's strategy, the player's sampling strategy, and the player's utility function.

D. Mixture of Categorical Distributions Maximum Likelihood

Consider a mixture of K categorical distributions with parameters $\vec{\mu} = (\vec{\mu}_1, \vec{\mu}_2, \ldots, \vec{\mu}_K)$ and $\vec{\pi} = (\pi_1, \pi_2, \ldots, \pi_K)$. Each $\vec{\mu}_k = (\mu_{k1}, \mu_{k2}, \ldots, \mu_{kD})$, where μ_{ki} is the probability of sampling category i from categorical distribution k, $\sum_{i=1}^{D} \mu_{ki} = 1$, $0 \leq \mu_{ki} \leq 1$ for all $1 \leq i \leq D$.

Each π_k is the probability of sampling categorical distribution k, $\sum_{k=1}^{K} \pi_k = 1$, $0 \leq \pi_k \leq 1$ for all $1 \leq k \leq K$. A categorical variable drawn from this mixture is a 1-of-D encoded vector $\vec{x} = (x_1, x_2, \ldots, x_D)$, where one component is 1 and the rest are 0. The probability of sampling $\vec{x}$ given $\vec{\mu}$ and $\vec{\pi}$ is

$$\Pr(\vec{x}|\vec{\mu}, \vec{\pi}) = \sum_{k=1}^{K} \pi_k \Pr(\vec{x}|\vec{\mu}_k). \tag{A.14}$$

Given a data set X of N samples from this mixture $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$. The likelihood of $\vec{\mu}$ and $\vec{\pi}$ given X is

$$L(\vec{\mu}, \vec{\pi}; X) = \Pr(X|\vec{\mu}, \vec{\pi}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \Pr(\vec{x}_n|\vec{\mu}_k). \tag{A.15}$$

The log-likelihood of $\vec{\mu}$ and $\vec{\pi}$ given X is

$$\ln L(\vec{\mu}, \vec{\pi}; X) = \ln \Pr(X|\vec{\mu}, \vec{\pi}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \Pr(\vec{x}_n|\vec{\mu}_k). \tag{A.16}$$

Since a summation is in the logarithm this does not have a closed-form solution, so we will derive expectation-maximisation equations for maximising this likelihood. For each $\vec{x}$ introduce a latent variable, which is a 1-of-K encoded vector $\vec{z} = (z_1, z_2, \ldots, z_K)$, where one component is 1 and the rest are 0, its value indicates which categorical distribution generated $\vec{x}$. The probability of $\vec{x}$ and $\vec{z}$ given $\vec{\mu}$ and $\vec{\pi}$ is

$$\Pr(\vec{x}, \vec{z}|\vec{\mu}, \vec{\pi}) = \prod_{k=1}^{K} \pi_k^{z_k} \Pr(\vec{x}|\vec{\mu}_k)^{z_k}. \tag{A.17}$$

The likelihood of $\vec{\mu}$ and $\vec{\pi}$ given X and Z is

$$\begin{aligned} &L(\vec{\mu}, \vec{\pi}; X, Z) = \Pr(X, Z|\vec{\mu}, \vec{\pi}) \\ &= \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \Pr(\vec{x}_n|\vec{\mu}_k)^{z_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \left(\prod_{i=1}^{D} \mu_{ki}^{x_{ni}} \right)^{z_{nk}}. \end{aligned} \tag{A.18}$$

The log-likelihood of $\vec{\mu}$ and $\vec{\pi}$ given X and Z is

$$\begin{aligned} \ln \Pr(X, Z|\vec{\mu}, \vec{\pi}) &= \ln \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \left(\prod_{i=1}^{D} \mu_{ki}^{x_{ni}} \right)^{z_{nk}} \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki} \right). \end{aligned} \tag{A.19}$$

Taking the expected value with respect to the posterior distribution of Z gives $\mathbb{E}_Z[\ln \Pr(X, Z|\vec{\mu}, \vec{\pi})] =$

$$\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})\left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki}\right), \tag{A.20}$$

where $\gamma(z_{nk}) = \mathbb{E}[z_{nk}]$ is the posterior probability, or responsibility, of categorical distribution k for sample $\vec{x}_n$. This is evaluated in the E-step as $\gamma(z_{nk}) = \mathbb{E}[z_{nk}] =$

$$\frac{\sum_{\vec{z}_n} z_{nk} \prod_{k'}[\pi_{k'} \Pr(\vec{x}_n|\vec{\mu}_{k'})]^{z_{nk'}}}{\sum_{\vec{z}_n} \prod_j [\pi_j \Pr(\vec{x}_n|\vec{\mu}_j)]^{z_{nj}}} = \frac{\pi_k \Pr(\vec{x}_n|\vec{\mu}_k)}{\sum_{j=1}^{K} \pi_j \Pr(\vec{x}_n|\vec{\mu}_j)}. \tag{A.21}$$

Using a Lagrange multiplier $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_K)$ to create a new function, which takes into account the constraint $\sum_{i=1}^{D} \mu_{ki} = 1$ for all $1 \leq k \leq K$, gives

$$\begin{aligned} G(\vec{\mu}, \vec{\pi}, \vec{\lambda}; X, Z) &= \mathbb{E}_Z[\ln \Pr(X, Z|\vec{\mu}, \vec{\pi}, \vec{\lambda})] \\ &= \left[\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})\left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki}\right)\right] \\ &\quad - \left[\sum_{k=1}^{K}\lambda_k\left[\left(\sum_{i=1}^{D}\mu_{ki}\right) - 1\right]\right]. \end{aligned} \tag{A.22}$$

Taking the partial derivatives of this function, firstly with respect to one probability μ_{dc}, and secondly with respect to one Lagrange multiplier component λ_d gives

$$\begin{aligned} &\frac{\partial}{\partial \mu_{dc}}\left(\left[\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})\left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki}\right)\right]\right. \\ &\left. - \left[\sum_{k=1}^{K}\lambda_k\left[\left(\sum_{i=1}^{D}\mu_{ki}\right) - 1\right]\right]\right) = \left[\sum_{n=1}^{N}\gamma(z_{nd})\frac{x_{nc}}{\mu_{dc}}\right] - \lambda_d, \end{aligned} \tag{A.23}$$

$$\begin{aligned} &\frac{\partial}{\partial \lambda_d}\left(\left[\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})\left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki}\right)\right]\right. \\ &\left. - \left[\sum_{k=1}^{K}\lambda_k\left[\left(\sum_{i=1}^{D}\mu_{ki}\right) - 1\right]\right]\right) = 1 - \sum_{i=1}^{D}\mu_{di}. \end{aligned} \tag{A.24}$$

To find the maximising parameters, we set the partial derivatives equal to zero, which gives

$$\lambda_d = \frac{1}{\mu_{dc}}\sum_{n=1}^{N}\gamma(z_{nd})x_{nc}, \tag{A.25}$$

$$\sum_{i=1}^{D}\mu_{di} = 1. \tag{A.26}$$

With some manipulations we can find μ_{dc} as follows

$$\sum_{i=1}^{D}\lambda_d\mu_{di} = \lambda_d\sum_{i=1}^{D}\mu_{di} = \lambda_d = \sum_{i=1}^{D}\sum_{n=1}^{N}\gamma(z_{nd})x_{ni},$$

$$\mu_{dc} = \frac{\sum_{n=1}^{N}\gamma(z_{nd})x_{nc}}{\sum_{i=1}^{D}\sum_{n=1}^{N}\gamma(z_{nd})x_{ni}}. \tag{A.27}$$

ACKNOWLEDGEMENTS

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/P505631/1] and the University of Manchester.

REFERENCES

[1] "The annual computer poker competition," http://www.computerpokercompetition.org/, accessed: 13/10/2014.

[2] V. P. Crawford, "Learning the optimal strategy in a zero-sum game," *Econometrica*, vol. 42, pp. 885–891, 1974.

[3] ——, "Learning behavior and mixed-strategy Nash equilibria," *JEBO*, vol. 6, pp. 69–78, 1985.

[4] ——, "Learning and mixed-strategy equilibria in evolutionary games," *JTB*, vol. 140, pp. 537–550, 1989.

[5] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *AI*, vol. 136, pp. 215–250, 2002.

[6] S. Abdallah and V. R. Lesser, "Non-linear dynamics in multiagent reinforcement learning algorithms," in *AAMAS*, 2008, pp. 1321–1324.

[7] C. Zhang and V. Lesser, "Multi-agent learning with policy prediction," in *AAAI*, 2010, pp. 927–934.

[8] M. Johanson *et al.*, "Computing robust counter-strategies," in *NIPS*. MIT Press, 2008, pp. 1128–1135.

[9] D. Billings *et al.*, "Opponent modeling in poker," in *AAAI*, 1998, pp. 493–499.

[10] A. Davidson *et al.*, "Improved opponent modeling in poker," in *ICAI*, 2000, pp. 1467–1473.

[11] M. Ponsen *et al.*, "Bayes-relational learning of opponent models from incomplete information in no-limit poker," in *AAAI*, 2008, pp. 1485–1486.

[12] G. Broeck *et al.*, "Monte-Carlo Tree Search in poker using expected reward distributions," in *ACML*, 2009, pp. 367–381.

[13] J. Rubin and I. Watson, "Similarity-based retrieval and solution re-use policies in the game of Texas hold'em," in *ICCBR*, 2010, pp. 465–479.

[14] K. Korb *et al.*, "Bayesian poker," in *UAI*, 1999, pp. 343–350.

[15] F. Southey *et al.*, "Bayes' bluff: Opponent modelling in poker," in *UAI*, 2005, pp. 550–558.

[16] R. Baker and P. Cowling, "Bayesian opponent modeling in a simple poker environment," in *CIG*, 2007, pp. 125–131.

[17] N. Risk and D. Szafron, "Using counterfactual regret minimization to create competitive multiplayer poker agents," in *AAMAS*, 2010, pp. 159–166.

[18] M. Johanson and M. Bowling, "Data biased robust counter strategies," in *AISTATS*, 2009, pp. 264–271.

[19] M. Ponsen *et al.*, "Computing approximate Nash equilibria and robust best-responses using sampling," *JAIR*, vol. 42, pp. 575–605, 2011.

[20] N. Bard *et al.*, "Online implicit agent modelling," in *AAMAS*, 2013, pp. 255–262.

[21] S. Ganzfried and T. Sandholm, "Game theory-based opponent modeling in large imperfect-information games," in *AAMAS*, 2011, pp. 533–540.

[22] T. Sandholm, "The state of solving large incomplete-information games, and application to poker," *AI Magazine*, vol. 31, no. 4, pp. 13–32, 2010.

[23] J. Rubin and I. Watson, "Computer poker: A review," *AI*, vol. 175, no. 5–6, pp. 958–987, 2011.

[24] M. Lanctot *et al.*, "No-regret learning in extensive-form games with imperfect recall," in *ICML*, 2012, pp. 65–72.

[25] J. Shi and M. L. Littman, "Abstraction methods for game theoretic poker," in *Revised Papers from CG 2*, 2000, pp. 333–345.

[26] M. Johanson, "Robust strategies and counter-strategies: Building a champion level computer poker player," Master's thesis, UOFA, 2007.

[27] M. Lanctot *et al.*, "Monte Carlo sampling for regret minimization in extensive games," in *NIPS*, 2009, pp. 1078–1086.

[28] P. Auer *et al.*, "Finite-time analysis of the multiarmed bandit problem," *ML*, vol. 47, no. 2–3, pp. 235–256, 2002.

[29] M. Zinkevich *et al.*, "Regret minimization in games with incomplete information," in *NIPS*, 2008, pp. 905–912.

[30] A. Dempster *et al.*, "Maximum likelihood from incomplete data via the EM algorithm," *JRSS*, vol. 39, pp. 1–38, 1977.

[31] P. Liang and D. Klein, "Online EM for unsupervised models," in *NAACL*, 2009, pp. 611–619.

[32] M.-A. Sato and S. Ishii, "On-line EM algorithm for the normalized Gaussian network," *Neural Computation*, vol. 12, pp. 407–432, 2000.

[33] O. Cappé and E. Moulines, "On-line Expectation-Maximization algorithm for latent data models," *JRSS*, vol. 71, no. 3, pp. 593–613, 2009.

[34] J. M. Butterworth, "Stability of gradient-based learning dynamics in two-agent imperfect-information games," PhD, 2010.

[35] S. Jensen *et al.*, "Non-stationary policy learning in 2-player zero sum games," in *AAAI*, 2005, pp. 789–794.

[36] S. Jensen, "Learning in dynamic temporal domains using contextual prediction entropy as a guiding principle," PhD, 2010.

[37] M. Lanctot, "Monte Carlo sampling and regret minimization for equilibrium computation and decision-making in large extensive form games," PhD, 2013.

[38] "Least-squares algorithms," http://www.mathworks.com/help/optim/ug/least-squares-model-fitting-algorithms.html, accessed: 13/10/2014.

Richard Mealing was born in Chester, U.K., in 1987. He received a B.Sc. degree (with honours) in Astrophysics and Computer Science, and an M.Sc. degree (with distinction) in Computer Science from the University of Liverpool, Liverpool, U.K., in 2009 and 2010 respectively. He received a Ph.D. degree in Computer Science from the University of Manchester, Manchester, U.K., in 2015. His main research interests include (with a particular focus on games) non-cooperative learning, dynamic opponent modelling, and finding equilibrium solutions. He is currently working in industry looking to apply his knowledge and gain experience.

Jonathan L. Shapiro received his Ph.D. degree in physics from the University of California at Los Angeles (UCLA), Los Angeles, in 1986. He is a Reader in Computer Science at the University of Manchester, Manchester, U.K., where he heads the Machine Learning and Optimisation Research Group. His current research is in reinforcement learning in dynamic environments and in games, probabilistic modelling, and theoretical approaches to evolutionary dynamics. Industrial applications include localisation, and anomaly detection in sensor networks.

LIST OF FIGURES

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

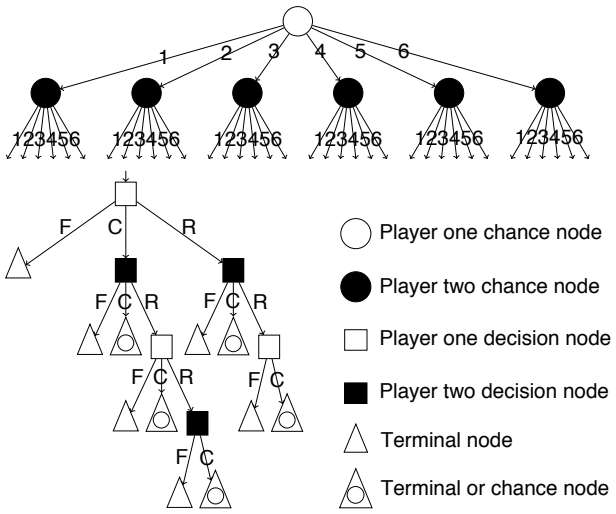


Fig. 1

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

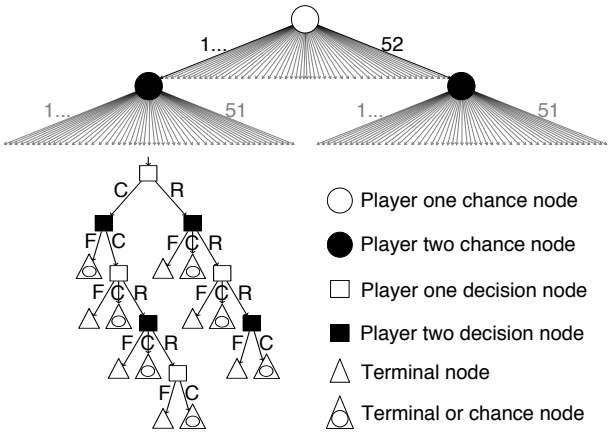


Fig. 2

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

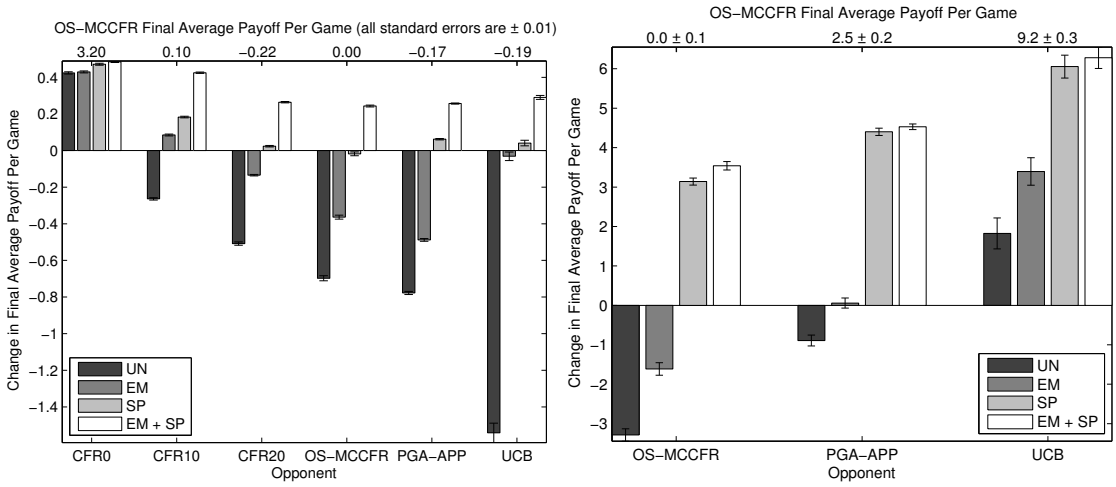
1: (1-7) Initialise a visit count per $I \in \mathcal{I}_{\text{opp}}$ and a sequence predictor per $I \in \mathcal{I}_{\text{opp}}$ where $P(I) = \text{opp}$
2: Play game vs opp, update strategy (OS-MCCFR), call OBSERVE
3: (10-14) E-step: for each terminal node (path) in your terminal information set, multiply opp's action probabilities (from the categorical distributions/visit counts) and the probability of its hidden information given your hidden information. Normalise these probabilities
4: (15-18) M-step: for each terminal node, increment visit counts of $I \in \mathcal{I}_{\text{opp}}$ along it by its E-step probability
5: (19) Sample h_{opp} according to E-step distribution
6: (20-22) Have each sequence predictor along path for h_{opp} observe opp's action taken in its information set
7: Simulate games vs model, update strategy (OS-MCCFR), call PREDICT (23-25) for its actions (using sequence predictors)
8: Repeat from 2 for a number of games vs the opponent

Fig. 3

Require: Player, Opponent information sets $\mathcal{I}_{\text{pla}}$, $\mathcal{I}_{\text{opp}}$, Lookback k
Initialise $M_v : \mathcal{I}_{\text{opp}} \to \mathbb{R}$ such that ▷ Start initialisation
$$M_v(I) \leftarrow \begin{cases} 1 & \text{if } I \subseteq Z \\ \sum_{a \in A(I)} M_v((I,a)) & \text{otherwise} \end{cases}$$
Initialise $M_{\text{pred}} : \{I : I \in \mathcal{I}_{\text{opp}}, P(I) = \text{opp}\} \to \mathcal{E}$
for all $I \in \text{dom}(M_{\text{pred}})$ **do**
  Initialise predictor p_I with k lookback $M_{\text{pred}}(I) \leftarrow p_I$
  for $i = 1$ to k **do**
    Sample action a with probability $\frac{M_v((I,a))}{M_v(I)}$
    Observe action a with $M_{\text{pred}}(I)$ ▷ End
function OBSERVE($(I' \subseteq Z) \in \mathcal{I}_{\text{pla}}$) ▷ Call after real game
  Initialise $M_t : \{I : I \in \mathcal{I}_{\text{opp}}, I \cap I' \neq \emptyset\} \to \mathbb{R}$
  for all $a_{1:m} \in \text{dom}(M_t)$ **do** ▷ Start E-step
    $M_t(a_{1:m} = (\mathcal{H}_{\text{opp}}, S)) \leftarrow \Pr(\mathcal{H}_{\text{opp}} | \mathcal{H}_{\text{pla}})$
    for $l \leftarrow 0$ to m **do**
      if $P(a_{1:l}) = \text{opp}$ **then**
        $M_t(a_{1:m}) \leftarrow M_t(a_{1:m}) \frac{M_v(a_{1:(l+1)})}{M_v(a_{1:l})}$ ▷ End
  for all $a_{1:m} \in \text{dom}(M_t)$ **do** ▷ Start M-step
    $M_t(I) \leftarrow \frac{M_t(I)}{\sum_{I'' \in \text{dom}(M_t)} M_t(I'')}$
    for $l \leftarrow 0$ to m **do** ▷ Update categoricals (visits)
      $M_v(a_{1:l}) \leftarrow M_v(a_{1:l}) + M_t(a_{1:m})$ ▷ End
  Sample $I = a_{1:m}$ with probability $M_t(I)$ ▷ Get h_{opp}
  for $l \leftarrow 0$ to m **do** ▷ Update sequence predictors
    if $P(a_{1:l}) = \text{opp}$ **then**
      Observe action a_{l+1} with $M_{\text{pred}}(a_{1:l})$
function PREDICT($h \in \{h : h \in H \setminus Z, P(h) = \text{opp}\}$) ▷ Call during simulated games
  Get I where $h \in I$ and $I \in \mathcal{I}_{\text{opp}}$
  return a prediction from $M_{\text{pred}}(I)$

Fig. 4

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games



(a) (b)

Fig. 5

LIST OF TABLES

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

TABLE I

Player	Parameters
CFRX	N/A
OS-MCCFR	explore rate = 0.05
PGA-APP	explore rate = 0.05, learning rate = 0.9, discount factor = 0.99, step-size = 0.01, prediction length = 1.0
UCB	constant = 3
OS-MCCFR OM	explore rate = 0.05, sequence predictor = ELPH (lookback = 5, entropy threshold = 0.1), games played against opponent model in-between games against the opponent = 100 in die-roll poker and 10 in Rhode Island hold'em
Other parameters	
number of games = 1×10^5, number of repeats = 80, both positions played per game	

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

TABLE II

(a)

Opponent model	c	$t_{0.99}$
None	3.2	1.2 $\times 10^5$
UN	3.6	6.2 $\times 10^4$
EM	3.6	6.4 $\times 10^4$
SP	3.7	5.3 $\times 10^4$
EM + SP	3.7	3.8 $\times 10^4$

(b)

c	$t_{0.99}$
0.15	2.4 $\times 10^5$
-0.12	2.3 $\times 10^5$
0.23	2.1 $\times 10^5$
0.30	1.7 $\times 10^5$
0.53	1.2 $\times 10^5$

(c)

c	$t_{0.99}$
-0.16	2.7 $\times 10^5$
-0.71	1.5 $\times 10^5$
-0.36	1.6 $\times 10^5$
-0.19	1.6 $\times 10^5$
0.05	1.9 $\times 10^5$

(d)

Opponent model	c	$t_{0.99}$
None	N/A	N/A
UN	-0.69	6.9 $\times 10^4$
EM	-0.41	2.1 $\times 10^4$
SP	-0.049	2.8 $\times 10^5$
EM + SP	0.22	1.9 $\times 10^5$

(e)

c	$t_{0.99}$
-0.18	7.4 $\times 10^4$
-0.96	7.2 $\times 10^4$
-0.71	3.5 $\times 10^4$
-0.1	1.1 $\times 10^5$
0.085	8.7 $\times 10^4$

(f)

c	$t_{0.99}$
-0.24	1.0 $\times 10^4$
-1.8	1.4 $\times 10^5$
-0.22	3.3 $\times 10^4$
-0.15	1.5 $\times 10^5$
0.10	1.4 $\times 10^5$

(g)

Opponent model	c	$t_{0.99}$
None	N/A	N/A
UN	-3.2	7.4 $\times 10^4$
EM	-1.4	5.6 $\times 10^4$
SP	2.9	2.0 $\times 10^5$
EM + SP	3.3	2.1 $\times 10^5$

(h)

c	$t_{0.99}$
1.8	2.8 $\times 10^5$
0.82	3.0 $\times 10^5$
1.9	2.5 $\times 10^5$
6.4	1.9 $\times 10^5$
6.1	2.2 $\times 10^5$

(i)

c	$t_{0.99}$
9.8	2.0 $\times 10^5$
12	1.7 $\times 10^5$
14	2.3 $\times 10^5$
16	1.5 $\times 10^5$
16	1.5 $\times 10^5$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2491611, IEEE Transactions on Computational Intelligence and AI in Games

TABLE III

Opponent	6 Sided	9 Sided	10 Sided
OS-MCCFR	0.26 ± 0.01	0.17 ± 0.01	0.12 ± 0.01
PGA-APP	0.19 ± 0.01	0.17 ± 0.01	0.09 ± 0.01
UCB	0.25 ± 0.02	0.24 ± 0.02	0.14 ± 0.02