Computing Equilibria in Multiplayer Stochastic Games of Imperfect Information *

Sam Ganzfried
Department of Computer Science
Carnegie Mellon University
sganzfri@cs.cmu.edu

Tuomas Sandholm
Department of Computer Science
Carnegie Mellon University
sandholm@cs.cmu.edu

Abstract

Computing a Nash equilibrium in multiplayer stochastic games is a notoriously difficult problem. Prior algorithms have been proven to converge in extremely limited settings and have only been tested on small problems. In contrast, we recently presented an algorithm for computing approximate jam/fold equilibrium strategies in a three-player no-limit Texas hold'em tournament—a very large real-world stochastic game of imperfect information [5]. In this paper we show that it is possible for that algorithm to converge to a non-equilibrium strategy profile. However, we develop an *ex post* procedure that determines exactly how much each player can gain by deviating from his strategy and confirm that the strategies computed in that paper actually do constitute an ϵ-equilibrium for a very small ϵ (0.5% of the tournament entry fee). Next, we develop several new algorithms for computing a Nash equilibrium in multiplayer stochastic games (with perfect or imperfect information) which can provably never converge to a non-equilibrium. Experiments show that one of these algorithms outperforms the original algorithm on the same poker tournament. In short, we present the first algorithms for provably computing an ϵ-equilibrium of a large stochastic game for small ϵ. Finally, we present an efficient algorithm that minimizes external regret in both the perfect and imperfect information cases.

1 Introduction

Poker exemplifies many challenging problems in computational game theory and multiagent systems: it has enormous strategy spaces, it is a game of imperfect information, it can have many players, and poker tournaments are stochastic games. There has been much interest recently in devising strong game-theoretic strategies for a single hand of two-player limit Texas hold'em (e.g., [1, 6, 15]). However, there has been relatively little work on the no-limit variant despite the fact that it is much more popular among humans. (In limit poker bets must be of a fixed size, while in no-limit poker players can bet any amount up to the amount of chips they have left.) A notable exception is a recent paper by Gilpin *et al.* [7]. However, that paper still focuses on playing a single hand rather than a tournament.

A significant deviation point from prior literature was a recent paper that computes an approximate equilibrium for both a single hand and a tournament of two-player no-limit Texas hold'em [12]. That paper studies a restricted set of strategies known as *jam/fold strategies* and shows that the equilibrium profile when both players are restricted to such strategies closely approximates an equilibrium of the unrestricted game. However, the results and the algorithms of that paper do not apply to more than two players. There often exists a large complexity gap between the difficulty of solving two and three-player zero-sum games; for example, two-player zero-sum matrix games can be solved in polynomial time by linear programming while solving three-player matrix games is PPAD-complete [3]. Furthermore, poker tournaments are stochastic games, and there are no known algorithms that are guaranteed to converge to an equilibrium in three-player stochastic games (even in the zero-sum case).

Stochastic games generalize Markov decision processes (MDPs) to the multiagent setting: at each state all agents choose an action, and the transition function now depends on the joint action profile. As in other classes of games, the standard solution concept is the Nash equilibrium. Prior algorithms are guaranteed to converge to an equilibrium in certain classes of games. For example, *Friend-or-Foe Q-learning* [11] and *Nash-Q* [8] converge if the overall stochastic game has a global optimum (a strategy profile that maximizes the payoff for each agent) or a saddle point (Nash equilibrium such that if any agent deviates, all others become better off)—and furthermore, the algorithm needs to know which of these two settings it is operating in. Other algorithms are guaranteed to converge to an optimal Nash equilibrium in team stochastic games—games in which all agents receive the same payoff [2, 14]. Interesting algorithmic results have also been proven for planning in general-sum stochastic games [10]. That paper does not specify a Nash selection function; we provide such a function that is effective in practice. Their algorithm *Finite-VI* resembles our algorithm *VI-FP* from [5], which we will review, but they do not have algo-

*This material is based upon work supported by the National Science Foundation under ITR grant IIS-0427858. We also acknowledge Intel Corporation and IBM for their machine gifts.

rithms resembling those that we will introduce in this paper. Furthermore, their algorithms find equilibria only in finite-horizon games; ours also do in infinite games. In summary, the settings of prior work are quite restrictive, and all of those algorithms have only been tested on tiny problems, if at all.

In contrast, our recent paper presents an algorithm for computing approximate jam/fold equilibrium strategies in a three-player no-limit Texas hold'em tournament [5]. It is easy to see that the tournament does not fall into the classes of games for which prior algorithms are guaranteed to converge, and furthermore the game is many orders of magnitude larger than the games previously solved. Additionally, the stage games have imperfect information: players are dealt private cards at each game state and must choose their action without knowing the cards dealt to the other players. The tournament uses the actual parameters of tournaments played on the most popular poker site, *Pokerstars*. As we showed, it can be formulated as a stochastic game with 946 states, each with $2^{6\times 169}$ possible actions. Despite the fact that our earlier algorithm converged when run on that tournament, very little is known about it. In particular, we show that it is possible for that algorithm to converge to a non-equilibrium. Thus, the strategy profile computed in our earlier paper was not guaranteed to be a jam/fold equilibrium of the tournament.

Fortunately, we show in this paper that the strategies computed by that algorithm actually do constitute an approximate jam/fold equilibrium of the tournament. We do so by developing an *ex post* checking procedure. The check reveals that no player can gain more than 0.5% of the tournament entry fee by deviating from the computed strategy for any starting stacks. The techniques we use in the *ex post* check also suggest new algorithms for solving stochastic games. We present several new algorithms with the guarantee that if the algorithm converges, the resulting strategy profile is an equilibrium. One of these algorithms outperforms our original algorithm on the same poker tournament. In short, this paper presents the first algorithms for provably computing an ϵ-equilibrium of a large stochastic game for small ϵ. Finally, we present an efficient algorithm that minimizes external regret in both the perfect and imperfect information case.

2 Rules of no-limit Texas hold'em

In this section we briefly review the rules of no-limit Texas hold'em. Each player at the table is dealt two private *hole cards*, and one player is selected to be the *button*—a designation which shifts one position clockwise each hand. The player to the left of the button is called the *small blind (SB)*, and the player to his left is the *big blind (BB)*. Both the SB and BB are forced to put some number of chips into the pot before the hand (normally BB = 2 × SB); these investments are called *blinds*. Then there is a round of betting starting with the player to the BB's left. After that, three cards (called the *flop*) are dealt face up in the middle of the table. Then there is another round of betting starting with the player directly left of the button, followed by another card dealt face up (the *turn*). Then another round of betting, followed by a fifth card face up (the *river*), followed by a final round of betting. If there is more than one player who has not folded, the player with the best five-card hand (constructed from his two hole cards and the five community cards) wins the pot. In case of a tie, the pot is split evenly among those players.

During each round of betting, each player has four possible options. (1) *fold*: pass and forfeit his chance of winning the pot. (2) *call*: put a number of chips equal to the size of the current bet into the pot. (3) *raise*: put some additional amount of chips in the pot beyond what was needed to make a call. (4) *jam* (also referred to as moving *all-in* or *pushing*): put all of one's remaining chips into the pot (even if that amount is smaller than the amount needed to call; in this case a *side pot* is created for the other players so that everyone has invested equally into the main pot).

3 Poker tournaments and stochastic games

Tournaments are an extremely popular form of poker; they work as follows. Some number of players (say 10) all pay an entry fee (say $10) and are given a number of chips (say 1500), which have no monetary value *per se* except that a player is eliminated from the tournament when he runs out of chips. The first seven players eliminated from the tournament lose their entry fee, while the three top finishers receive 50%, 30%, and 20% of the sum of the entry fees ($100) respectively. Usually the blinds increase every five or ten minutes in online tournaments, starting at SB = 10, BB = 20 and approximately doubling at every level. Since chips have no explicit monetary value, tournaments are actually stochastic games, where each *state* corresponds to a vector of stack sizes.

Formally, a stochastic game G is a tuple (N, S, A, p, r) where $N = \{1, \ldots, n\}$ denotes a finite set of players, and S denotes a finite set of states. A is a tuple $(A_1, \ldots, A_n)$ where for each $i \in N$, $s \in S$, $A_i(s)$ is a finite set corresponding to player i's available actions at state s. For $s \in S$ let $A(s)$ denote the vector $(A_1(s), \ldots, A_n(s))$. For each $s, t \in S$ and $a \in A(s)$, $p_{s,t}(a)$ denotes the probability that we transition from state s to state t when all players play their component of the action vector a. Finally, $r : S \to \mathbb{R}^n$ denotes the payoff function, where for each $s \in S$, $r(s)$ is a vector whose i'th component is the payoff to player i when state s is reached (in some formulations the reward function has domain $S \times A$).

At each state s, player i can choose any probability distribution σ over actions in $A_i(s)$. Let $\sigma(a_i)$ denote the probability he selects action a_i. Let $a = (a_1, \ldots, a_n)$, and define $\sigma(a) = \prod_i \sigma(a_i)$. Then we extend our definition of the transition function so that $p_{s,t}(\sigma) = \sum_{a \in A(s)} [\sigma(a) p_{s,t}(a)]$. A *policy* π for player i is a choice of probability distributions over actions in $A_i(s)$ at each state s.

If we let s_t denote the current state at time t then the goal of agent i is to maximize $\sum_{t=0}^{\infty} \gamma^t r_i(s_t)$, where $0 < \gamma \leq 1$ is the discount rate. If $\gamma = 1$ then we say that the game is *undiscounted*, and otherwise we say that it is *discounted*.

We say that a state s is *terminal* if $p_{s,t}(a) = 0$ for all $t \neq s, a \in A(s)$ and $p_{s,s}(a) = 1$ for all $a \in A(s)$. If we ever arrive at a terminal state s, then we will remain at s for the remainder of the game. A state is *nonterminal* if it is not a terminal state.

A stochastic game with one agent is called a *Markov decision process (MDP)*.

4 Existing algorithm for three-player tournaments (*VI-FP*)

When blinds become sufficiently large relative to stack sizes in tournaments, rationality dictates more aggressive play. In particular, prior work suggests that it is near optimal for players to either go all-in or fold preflop [5, 12] (such strategies are called *jam/fold strategies*). In this section we review our recent algorithm for computing an approximate jam/fold equilibrium in a three-player tournament [5]. The algorithm consists of iteration at two nested levels. The "inner loop" uses an extension of smoothed fictitious play to determine ϵ-equilibrium strategies for playing a hand at a given state (stack vector) given the values of possible future states. This is done for all states. The iteration of the "outer loop" adjusts the values of the different states in light of the new payoffs obtained from the inner loop. Then the inner loop is executed again until convergence, then the outer loop, and so on. The algorithm terminates when no state's payoff changes by more than δ between outer loop iterations. As the outer loop resembles Value Iteration and the inner loop uses smoothed Fictitious Play, we refer to this algorithm as *VI-FP*.

4.1 Inner loop

In standard fictitious play, each player plays a best response to the average strategies of his opponents thus far. Similarly, in smoothed fictitious play each player i applies the following update rule at each time step t to obtain his current strategy:

$$s_i^t = \left(1 - \frac{1}{t}\right) s_i^{t-1} + \frac{1}{t} s_i'^t, \qquad (1)$$

where $s_i'^t$ is a best response of player i to the profile s_{-i}^{t-1} of the other players played at time $t-1$ (strategies can be initialized arbitrarily at $t = 0$). This algorithm was originally developed as a simple learning model for repeated games, and was proven to converge to a Nash equilibrium in two-player zero-sum games [4]. However, it is not guaranteed to converge in two-player general-sum games or games with more than two players.

In poker tournaments the strategy spaces are exponential in the number of possible private signals. In particular, in Texas hold'em there are 169 strategically distinct preflop hands (see, e.g., [6]) and two jam/fold moves each player can make at each information set. Therefore, the strategy spaces (for any given stack vector) are of size 2^{169}, $2^{2\times 169}$, and $2^{3\times 169}$ for the button, small blind, and big blind, respectively (the big blind does not need to act when the other two players fold). To deal efficiently with this exponential blowup, the algorithm works with the extensive form of the game instead of enumerating the strategies. In the extensive form, each player's best response can be computed by traversing his information sets one at a time. The button has 169 information sets, the small blind has 2×169, and the big blind has 3×169. Therefore, the best response (for each player in turn) can be computed efficiently.

While fictitious play is not guaranteed to converge, we fortunately have the following result:

Theorem 1. *(Fudenberg and Levine [4]) Under fictitious play, if the empirical distributions over each player's choices converge, the strategy profile corresponding to the product of these distributions is a Nash equilibrium.*

4.2 Outer loop

As stated earlier, poker tournaments are stochastic games in which each state corresponds to a vector of stack sizes. In particular, we analyze the tournament in which there are 13500 total chips, and blinds are SB = 300, BB = 600. These correspond to common endgame parameters of sit-and-go tournaments at *Pokerstars*, the most popular online poker site. Each state in our game G is defined by a triple (x_1, x_2, x_3), where x_1 denotes the stack of the button, x_2 denotes the stack of the small blind, x_3 denotes the stack of the big blind, $\sum_i x_i = 13500$, and all x_i are multiples of 300. When one of the x_i becomes zero, that player is eliminated, and we can already solve the remaining game with the prior techniques because there are at most two players left. So we can ignore states in which stacks are zero from our model and just substitute the corresponding payoffs to players when we arrive in such states. G has 946 non-terminal states.

The algorithm for solving the game works as follows. First we initialize the assignment V^0 of payoffs to each player at each game state using a heuristic from the poker community known as the Independent Chip Model (ICM), which asserts that a player's probability of winning is equal to his fraction of all the chips; his probability of coming in second is asserted to be the fraction of remaining chips conditional on each other player coming in first, etc. [5]. ICM has been shown to be quite accurate when two players remain in the tournament [12]; when three players remain it has been shown to be relatively accurate overall, although in some states it is significantly inaccurate [5].

Next, suppose we start at some game state x^0. If we assume that the payoffs of V^0 at all states are the actual values of those states to the players, then the whole stochastic game just becomes a standard game in which every transition from x^0 leads to a terminal payoff. So we can run the fictitious play algorithm described in Section 4.1 at state x^0 until it (hopefully) converges. Supposing it does converge to an approximate equilibrium, each player will now have some new expected payoff at state x^0 if that equilibrium is played. (These payoffs might differ from V^0.) If we do this for all 946 game states, we come up with a new assignment V^1 of payoffs to each player at each game state. Now suppose we repeat this process and that hypothetically the payoffs remain the same between iterations k and $k+1$ (all the payoffs in V^k and V^{k+1} are equal). This would suggest that the strategies computed by fictitious play at iteration k (assuming they converge) are close to an equilibrium of the full game G.

Now, stated explicitly with tolerances, the overall algorithm works as follows. First, fix $\gamma, \delta > 0$. Initialize the V^0 to ICM, and run fictitious play using V^0 as payoffs until an γ-equilibrium is reached (a strategy profile in which no player can gain more than γ in expectation by deviating). Then use the payoffs V^1 which result from the computed strategy profile and repeat. The algorithm halts when it reaches an iteration k such that each payoff of V^k differs from the corresponding payoff of V^{k-1} by less than δ.

Algorithm 1 *VI-FP*(γ, δ)

```
V^0 = initializeValues()
diff = ∞
i = 0
while diff > δ do
  i = i + 1
  regret = ∞
  S = initializeStrategies()
  while regret > γ do
    S = fictPlay()
    regret = maxRegret(S)
  end while
  V^i = getNewValues(V^{i-1},S)
  diff = maxDev(V^i, V^{i-1})
end while
return S
```

5 Our *ex post* check

We now move to the contribution of this paper. In the experimental results with *VI-FP*, both the inner and outer loops converged. However, we observe that this does not guarantee that the final strategies constitute an approximate equilibrium. For example, suppose we initialized all the payoffs at nonterminal states to be \$100 (higher than the first-place payoff) and initialized payoffs at the terminal states using ICM. Then the optimal strategy for each player would be to fold every hand (except for the shortest stack if he is all-in), and the algorithm would converge to this profile (in a single outer-loop iteration). However, this profile is clearly not an equilibrium of the game, as the players will not receive any payoff (while they would receive at least \$20 if the game ended after some finite time). Thus there exist initializations for which *VI-FP* converges to a non-equilibrium profile.

Despite this fact, we suspected that the computed strategies formed an approximate equilibrium (as the strategies seemed very reasonable and we used an intelligent initialization). To verify this, we developed an *ex post* checking procedure to make sure. It computes the best response of each player to the computed strategy profile and determines the improvement in payoff (while the overall algorithm is somewhat obvious, it is difficult in our setting for reasons described below). If the improvement is less than ϵ for each player, then the original strategies indeed constitute an ϵ-equilibrium.

Stepping back to the big picture, *VI-FP* is an incomplete algorithm; but if it converges, our *ex post* check can be used to check the quality of the solution. Furthermore, the check can be applied at every iteration of the outer loop, even if *VI-FP* does not converge. As we will see, we have developed several new algorithms for solving stochastic games that cannot converge to a non-equilibrium strategy profile (as *VI-FP* can). Thus, if the new algorithms converge we will not need to waste extra resources performing the *ex post* check after every iteration, as we are guaranteed to be at an equilibrium.

We will now describe the *ex post* check in detail. The first step is to construct the MDP induced by the strategy profile s^* (computed by *VI-FP*). Denote this MDP by M. Each nonterminal state s_i of M is determined by a nonterminal state v_i of the stochastic game G (stack vector) and a position α_i (button, small blind, or big blind). So, M contains $946 \times 3 = 2838$ nonterminal states. In addition, M contains a terminal state for each terminal of G corresponding to each stack vector with at most two nonzero stacks that is reachable from a nonterminal state (say that a state s_2 is *reachable* from state s_1 if there exists an $a \in A(s_1)$ such that $p_{s_1,s_2}(a) > 0$).

The set of actions available at each state is essentially the set of jam/fold strategies available at that stack vector in G. If α_i is the button then there are 2^{169} possible actions at state s_i; if α_i is the small blind there are $2^{2\times169}$ actions; and if α_i is the big blind there are $2^{3\times169}$ actions. The transitions of M are determined by the probability distribution over states when player α_i chooses action a_i and the other players follow the strategy profile s^*_{-i}. It is important to note that in a transition to a nonterminal state the position changes; that is, if we are at state s_i with position α_i equal to the button and transition to state s_j, the new position α_j is the big blind (because blinds move clockwise). Finally, the rewards at each nonterminal state are 0, and at each terminal state they are the ICM payoff of the corresponding stack vector (since previous two-player results showed that ICM payoffs very closely approximate equilibrium payoffs in this case).

An optimal policy for M corresponds to a best response of each player in G to the profile s^*. However, we cannot apply the standard value or policy iteration algorithms for solving the MDP because we are in the undiscounted setting (since the players only care about their final payoffs). Instead we turn to variants of these algorithms that work when the objective is expected total reward and the following conditions hold: 1) for all states s and policies π, the value of state s under π is finite and 2) for each s there exists at least one available action a that gives nonnegative reward. In the results below, we refer to these conditions as "our setting."

For value iteration, the change from the classic discounted setting is the initialization:

Theorem 2. *(Puterman [13]) In our setting, if v^0 is initialized pessimistically (i.e., $\forall s, v^0(s) \leq v^*(s)$), value iteration converges (pointwise and monotonically) to v^*.*

5.1 Policy iteration

We refer the reader to [13] for a thorough description of the standard policy iteration algorithm and present the modified algorithm needed to obtain convergence in our setting here (while noting deviations from the classic setting). Policy iteration converges to an optimal policy in our setting if 1) our initial policy obtains non-negative expected total reward (which is obviously the case here), 2) we choose the minimal non-negative solution of the system of equations in the evaluation step (if there is a choice), and 3) if the action chosen for some state in the previous iteration is still among the optimal actions for that state, then select it again.

Since step 2 is critical for our new algorithms for solving multiplayer stochastic games in the next section, we will explain it in more detail here. For each state i in the MDP M, $v(i)$ is a variable corresponding to its value. Thus, the equation in step 2 is linear in the unknowns $v(i)$, $i = 1, \ldots, |S|$, where $|S|$ is the number of states in M. Thus we have a linear system of $|S|$ equations with $|S|$ unknowns. If the equations

Algorithm 2 Policy iteration for our setting

1. Set $n = 0$ and initialize the policy π^0 so it has nonnegative expected reward.
2. Let v^n be the solution to the system of equations
$$v(i) = r(i) + \sum_j p_{ij}^{\pi^n} v(j)$$
where $p_{ij}^{\pi^n}$ is the probability of moving from state i to state j under policy π^n. If there are multiple solutions, let v^n be the minimal nonnegative solution.
3. For each state s with action space $A(s)$, set
$$\pi^{n+1}(s) \in \operatorname*{argmax}_{a \in A(s)} \sum_j p_{ij}^a v^n(j),$$
breaking ties so that $\pi^{n+1}(s) = \pi^n(s)$ whenever possible.
4. If $\pi^{n+1}(s) = \pi^n(s)$ for all s, stop and set $\pi^* = \pi^n$. Otherwise increment n by 1 and return to Step 2.

are linearly independent, then there will be a unique solution which can be computed by matrix inversion. If there are no solutions, then we are in a degenerate case and the MDP has no optimal policy (it can be proven that this can never be the case in our setting). If there are multiple solutions, then we select the minimal nonnegative solution (see [13] for a further explanation). In all of our experiments the coefficient matrix had full rank and hence the system had a unique solution.

Theorem 3. *(Puterman [13]) Let S be the set of states in M. Suppose S and $A(s)$ are finite. Then, for some finite N, $v^N = v^*$ and $\pi^N = \pi^*$.*

5.2 *Ex post* check algorithm

Since we already have a policy s^* that we expect to be near-optimal (and the corresponding values from the final iteration of the outer loop) from *VI-FP*, we would like to use these values to obtain a warm start for v^0 in the *ex post* check. However, this would not work with the value iteration algorithm because v^0 would not be simultaneously pessimistic for each player (at any state) because we are in a zero-sum game (unless the values from *VI-FP* were precisely correct). On the other hand, we can use s^* as the initial policy in policy iteration because it clearly obtains non-negative expected total reward. Therefore we opt for using the warm start and choose policy iteration in the *ex post* check.

Algorithm 3 *Ex post* check

Create MDP M from the strategy profile s^*.
Run Algorithm 2 on M (using initial policy $\pi^0 = s^*$) to get π^*.
return $\max_{i \in N, s \in S} u_i(\pi_i^*(s), s^*_{-i}(s)) - u_i(s_i^*(s), s^*_{-i}(s))$

Proposition 1. *Algorithm 3 correctly computes the largest amount any agent can improve its expected utility by deviating from s^*.*

Proof. By Theorem 3, the policy iteration step of the *ex post* check converges in finite time to an optimal policy of the MDP M determined by s^*. Since the agent assumes that all other players are following s^* at each state of M, the final policy π^* is a best response to s^*. Thus the algorithm returns the largest amount any agent can gain by deviating from s^*. □

As in *VI-FP*, for efficiency in Step 3 of the policy iteration (Algorithm 2) we do not compute best responses in the exponential strategy space directly, but instead we consider the extensive form and compute best responses by traversing each agent's information sets in turn. Again, this avoids the exponential blowup.

We executed the *ex post* check on the strategies output by *VI-FP*. It converged in just two iterations. The conclusion from that experiment is that for any starting state of the tournament, no player can gain more than \$0.049 by deviating from s^*. This represents less than 0.5% of the tournament entry fee. In other words, while *VI-FP* does not guarantee that convergence is to an equilibrium, it happened to indeed converge to a strategy profile that is an ϵ-equilibrium for very small ϵ.

6 New algorithms for solving stochastic games

The approaches used in the *ex post* check motivate new algorithms for equilibrium finding as well. We now present those new algorithms. They apply to multiplayer stochastic games of either perfect or imperfect information in which best responses can be computed efficiently. Unlike *VI-FP*, each of the new algorithms has the following guarantee: if the algorithm converges, the final strategy profile is an equilibrium.

6.1 *PI-FP*: Policy Iteration as outer loop; Fictitious Play as inner loop

Our first new algorithm is similar to *VI-FP* except now the value updates take the form of Step 2 of Algorithm 2. After the inner loop of fictitious play converges at each stack vector to an ϵ-equilibrium s' (assuming the currently outer-loop values), we evaluate the expected payoffs at all stack vectors under the profile s' by constructing the induced MDP and finding the minimal solution of the system of equations determined by Step 2 of Algorithm 2. (We initialize the payoffs using ICM.) Thus, *PI-FP* differs from *VI-FP* in how the values are updated in the outer loop, while the inner loop remains the same.

Proposition 2. *If the sequence of strategies $\{s^n\}$ determined by iterations of the outer loop of this algorithm converges, then the final strategy profile s^* is an equilibrium.*

Proof. Let M denote the induced MDP determined by the profile s^* and define π^0 to be the policy such that $\pi^0(s) = s_i^*(s)$ for all states s in M, where i is the player corresponding to state s. Now suppose we run Algorithm 2 on M using π^0 as our initial policy. By the convergence of *PI-FP*, we must have $\pi^1(s) = \pi^0(s)$ for all s. Thus Algorithm 2 converges in one iteration on M, and we have $\pi^* = \pi^0 = s_i^*$. So, all players are playing a best response to each other under profile s^*, i.e., s^* is an equilibrium. □

Algorithm 4 *PI-FP*(γ, δ)

$V^0 = \text{initializeValues}()$
diff $= \infty$
$i = 0$
while diff $> \delta$ **do**
  $i = i + 1$
  regret = ∞
  S^0 = initializeStrategies()
  while regret $> \gamma$ **do**
    S^i = fictPlay()
    regret = maxRegret(S^i)
  end while
  $M^i = \text{createTransitionMatrix}(S^i)$
  $V^i = \text{evaluatePolicy}(M^i)$
  diff = maxDev(V^i, V^{i-1})
end while
return S^i

While *VI-FP* might converge to a non-equilibrium given a poor initialization (e.g., if all values are initialized to be above the first-place payoff), *PI-FP* can still recover from a poor initialization since it uses the values resulting from evaluating the policy, not the values from the initialization.

6.2 *FP-MDP*: switching the roles of Fictitious Play and MDP solving

Our next new algorithm involves reversing the roles of the inner and outer loops from the previous two algorithms: now fictitious play serves as the outer loop and MDP solving as the inner loop. As argued previously, we prefer policy iteration to value iteration for this application because it allows us to get a good warm start more easily (although value iteration may be preferable in some applications). (We initialize values using ICM and generate our initial strategy profile s^0 as before.) As in the *ex post* check, we construct the MDP induced from the strategy profile s^0 and compute the best response for each player using policy iteration. We combine the computed best responses with s^0 using the fictitious play updating rule to obtain s^1, and repeat until the sequence $\{s^n\}$ (hopefully) converges. We can use several different termination criteria, such as specifying a number of outer loop iterations or specifying a minimum deviation for the values or strategies between two outer loop iterations.

Algorithm 5 *FP-MDP*

S^0 = initializeStrategies()
$i = 0$
while termination criterion not met **do**
  M^i = constructMDP(S^i)
  S' = solveMDP(M^i)
  $S^{i+1} = \frac{i}{i+1} S^i + \frac{1}{i+1} S'$
  $i = i + 1$
end while
return S^i

Proposition 3. *If the sequence of strategies $\{s^n\}$ determined by iterations of the outer loop of this algorithm converges, then the final strategy profile s^* is an equilibrium.*

Proof. By Theorem 3, policy iteration will converge to an optimal (deterministic) policy at each step t of the inner loop, which corresponds to a best response to the strategy profile s_{-i}^{t-1}. Thus by Theorem 1, convergence of the outer loop implies we are at an equilibrium. □

Again, unlike *VI-FP*, *FP-MDP* can potentially recover from poor initializations because it only uses the values resulting from evaluating the policy.

6.3 *FTPL-MDP*: a polynomial time algorithm for regret minimization

Note that we could use any suitable MDP-solving algorithm in the inner loop of *FP-MDP*. In particular there exists a linear programming (LP) formulation for our setting [13]. We can treat this as a polynomial-time best-response oracle. If we replace fictitious play in the outer loop of *FP-MDP* with the follow-the-perturbed-leader (FTPL) algorithm [9], we obtain an efficient procedure for minimizing external regret in large multiplayer stochastic games. FTPL is similar to fictitious play except that instead of playing a best response to the profile s_{-i}^{t-1} at time t, player i picks a strategy j that maximizes $u_i(j, s_{-i}^{t-1}) + Z_{j,t}$, where $Z_{j,t}$ corresponds to random noise. The FTPL algorithm minimizes external regret; that is, if the game is repeated n times and a player follows the algorithm, then the per-period difference between the expected payoff of his best fixed strategy in hindsight and his expected payoff by following the FTPL algorithm approaches 0 (as $\frac{1}{\sqrt{n}}$).

Algorithm 6 *FTPL-MDP*

S^0 = initializeStrategies()
$i = 0$
while termination criterion not met **do**
  $\hat{S}^i$ = randomPerturbation(S^i)
  M^i = constructMDP($\hat{S}^i$)
  S' = solveMDP-LP(M^i)
  $S^{i+1} = \frac{i}{i+1} S^i + \frac{1}{i+1} S'$
  $i = i + 1$
end while
return S^i

7 Experiments

We conducted experiments comparing the performance of *VI-FP* with *PI-FP* and *FP-MDP* on the tournament using the payouts \$50, \$30, and \$20. The results are shown in Figure 1. The x-axis is the running time (in minutes, using 16 3GHz processors), and the y-axis is the maximal amount (in dollars) a player could improve by deviating from the current outer loop strategy profile of the algorithm (i.e., the ϵ of ϵ-equilibrium). The y-axis value is the maximum ϵ over all players and over all possible starting stack vectors of the tournament. The solid line denotes *VI-FP*, the dashed line denotes *PI-FP*, and the dashed/dotted line denotes *FP-MDP*.

The graph was generated by running the *ex post* check algorithm over the strategies computed at the end of each outer-loop iteration of the algorithms.

Since the outer loops for the algorithms take different amounts of time, the graph contains different numbers of data points for the different algorithms (the outer loop for *VI-FP* and *PI-FP* takes about 2 hours while it takes about 40 minutes for *FP-MDP*). We halted *VI-FP* and *PI-FP* when ϵ fell below 0.05, which represents 0.1% of the first place payoff; we halted *FP-MDP* after 20 hours, as it failed to achieve $\epsilon = 0.05$.

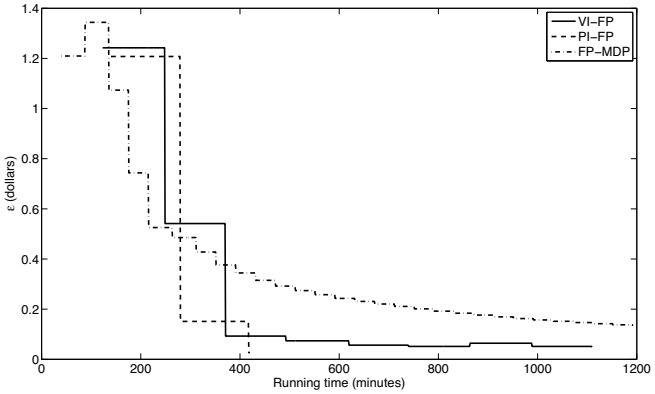

Figure 1: *Epsilon as a function of running time.*

PI-FP was the fastest to reach the target accuracy. *VI-FP* also reached it, but took almost three times as long. *FP-MDP* did not reach that accuracy in 20 hours.

8 Conclusions

Computing a Nash equilibrium in multiplayer stochastic games is a notoriously difficult problem. We pointed out that the best prior algorithm could converge to a non-equilibrium strategy profile; however, we developed an *ex post* check procedure that confirmed that the strategy profile computed for a large three-player poker tournament actually constitutes an ϵ-equilibrium for very small ϵ. We also presented several new algorithms for solving multiplayer stochastic games of imperfect information and proved that they can never converge to a non-equilibrium. Experiments on the poker tournament showed that one of the new algorithms (*PI-FP*) outperforms the prior algorithm *VI-FP*; thus the equilibrium guarantee comes at no cost in run time. Finally, we presented a polynomial-time algorithm for minimizing external regret in the same class of games.

We find it interesting that the algorithms converged to an equilibrium consistently and quickly despite the fact that they are not guaranteed to do so. None of the algorithms are guaranteed to converge at all, and *VI-FP* was not even guaranteed to be at an equilibrium even if it did converge. Hopefully the results of this paper could lead to the investigation of more general settings under which these convergence properties can be proven. While fictitious play is not guaranteed to converge in three-player games, it converged in every iteration of all of our algorithms; perhaps there is a more general class of multiplayer games (that includes poker tournaments) for which fictitious play is guaranteed to converge. Furthermore, the value iteration of the outer loop of *VI-FP* converged despite the fact that the initializations were clearly not pessimistic for every player; perhaps the assumptions of Theorem 2 can be weakened to allow limited optimistic initializations in some settings.

References

[1] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. *IJCAI*, 2003.

[2] R. Brafman and M. Tennenholtz. Learning to coordinate efficiently: A model-based approach. *JAIR*, 2003.

[3] X. Chen and X. Deng. Settling the complexity of 2-player Nash equilibrium. *FOCS*, 2006.

[4] D. Fudenberg and D. Levine. *The Theory of Learning in Games*. MIT Press, 1998.

[5] S. Ganzfried and T. Sandholm. Computing an approximate jam/fold equilibrium for 3-agent no-limit Texas hold'em tournaments. *AAMAS*, 2008.

[6] A. Gilpin and T. Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. *AAAI*, 2006.

[7] A. Gilpin, T. Sandholm, and T. B. Sørensen. A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. *AAMAS*, 2008.

[8] J. Hu and M. Wellman. Nash Q-learning for general-sum stochastic games. *JMLR*, 2003.

[9] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 2005.

[10] M. Kearns, Y. Mansour, and S. Singh. Fast planning in stochastic games. *UAI*, 2000.

[11] M. Littman. Friend-or-foe Q-learning in general-sum Markov games. *ICML*, 2001.

[12] P. B. Miltersen and T. B. Sørensen. A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament. *AAMAS*, 2007.

[13] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley & Sons, 2005.

[14] X. Wang and T. Sandholm. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. *NIPS*, 2002.

[15] M. Zinkevich, M. Bowling, M. Johanson, and C. Piccione. Regret minimization in games with incomplete information. *NIPS*, 2007.