Novel Approaches for Exploiting Opponents in Poker

Wesley Tansey
University of Texas at Austin
1 Inner Campus Drive
Austin, TX 78712 USA
tansey@cs.utexas.edu

ABSTRACT

Effectively exploiting opponents in incomplete information, extensive-form games is a challenging task for an online learning agent. Previous work has focused on either maintaining an explicit generative model that is updated directly based on observed opponent actions or implicit discriminative modeling via a portfolio of methods.

This paper introduces four new approaches to playing exploitive poker. First, a one-step temporal differencing version of counterfactual regret minimization called TD-CFR is presented. Second, an alternative implicit modeling approach based on the notion of subpolicies is explored. Third, a hybrid implicit and explicit algorithm is described that leverages a portfolio of models to bootstrap an explicit model. Finally, a combination of the second and third models is discussed along with an approach automatically deriving subpolicies from a portfolio of complete policies.

The game of Leduc poker is used as a benchmark domain to compare the two standard and four novel exploitive approaches against three different types of stationary opponents. Results from these experiments indicate that implicit modeling is a surprisingly robust approach in Leduc, outperforming the other five approaches in all three experiments. Additionally, policy bootstrapping appears to be effective at speeding up explicit modeling to be nearly as fast as implicit modeling, combining the benefits of both approaches at only a slight cost.

General Terms

Algorithms, Experimentation

Keywords

poker, opponent modeling, online learning

1. INTRODUCTION

Computer poker has been an active area of multiagent learning research in recent years. Most of this research area can be categorized as either optimal agent research, whose goal is to discover equilibria that provide lower bounds on payoffs, or maximal agent research, where agents try to model and exploit weaknesses in their opponents. While optimal agent research has produced a flurry of exciting results (e.g., [7, 6, 17, 12, 9]), research into exploitive agents has made comparatively less progress. To date, maximal agent research has focused on in-depth analyses of simple Bayesian techniques [14, 15], reducing the risk of model error [11, 10], and scaling portfolio response methods to full-scale poker [2]. Thus, there is a clear opportunity for improved approaches to opponent modeling and exploitation in poker.

To this end, this paper presents four novel approaches aimed at improving the sample efficiency of exploitive agents. We first extend a version of Counterfactual Regret Minimization (CFR) [17] from a Monte Carlo version previously introduced [12] to a temporal difference learning [16] context, in an algorithm we call *TD-CFR*. Next, we turn to Bayesian methods and introduce *policy bootstrapping*, an approach that leverages a portfolio of precomputed opponent models to bootstrap an explicit generative model. Third, we develop *subpolicy implicit modeling* based on the intuition that weaknesses in opponents occur in a systematic manner that creates correlations among different information sets. Finally, we combine the last two methods to create *subpolicy bootstrapping*, an approach that has the potential to flexibly increase sample efficiency even against opponents that look vastly dissimilar to those in its portfolio.

We conduct a series of experiments in the game of Leduc poker to compare each of the four novel methods with standard implicit and explicit modeling approaches from the literature. The results of our experiments suggest that policy bootstrapping is effective at speeding up explicit modeling, while implicit modeling remains the strongest approach. We examine the failure of implicit subpolicy modeling to improve on baseline implicit modeling and determine that Leduc may be too small of a game to benefit from such fine-grained modeling, even when systematic biases are hand-coded into opponents. We thus plan to run future experiments on larger games where subpolicies may be more useful.

2. BACKGROUND

In this section, we review some of the relevant prior work in poker agent research. For a detailed discussion on the rules and difficulty of the poker domain see [3].

2.1 Counterfactual Regret Minimization

The CFR algorithm [17] is an iterative algorithm that extends regret matching [8] to incomplete-information, extensive-form games to provably walk a strategy profile towards a cor-

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*
Copyright © 2013, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

related ϵ-equilibrium.[1] Later work has introduced speedups to CFR in the form of game tree compression [9] and Monte Carlo variants [12]. Among the MC-CFR implementations, Outcome Sampling CFR (OS-CFR) can be used for online regret minimization. The TD-CFR approach presented in this paper is based on OS-CFR.

2.2 Explicit opponent modeling

Explicit modeling of an opponent is perhaps the most natural way to conceive of an exploitive strategy. A probability distribution is estimated at each information set in the opponent's incomplete information gametree, with the results being updated after every observed outcome. Given the explicit model, a best response can be calculated and played. Data-Biased Responses (DBRs) [10] take a frequentist approach to explicit modeling, leveraging a large database of perfect information hands of opponent actions to build a model of opponent play. Once the opponent model is derived, DBRs compute a Restricted Nash Response (RNR) [11] that enables the agent to trade off exploitation and exploitability. While the resulting model is highly effective, the required large number of perfect-information hands and the sensitivity to the source of the samples makes DBRs impractical for short-term opponent exploitation.

An alternative, Bayesian approach to explicit modeling is to maintain a dirichlet prior at every information set [14, 15]. After every revealed hand, priors can then be updated; after a folded hand, the priors can be updated by marginalizing out the unobserved holecards. An exact opponent strategy can then be sampled via importance sampling or taken as the *maximum a posteriori* (MAP) strategy. Importance sampling is the more robust strategy since MAP inference may be a poor approximation if the distribution is multimodal. This paper uses importance sampling over a set of dirichlet priors, initialized to have a mode at the Nash equilibrium, for its explicit opponent modeling agent.

2.3 Implicit modeling

For large games, simple explicit modeling of an opponent may be impractical since the number of samples observed will generally cover only a small fraction of total game tree. Implicit opponent modeling [15] instead assumes the opponent is using a strategy drawn from some portfolio of potential strategies that is made available to the agent. After each outcome, the agent updates the likelihood of the opponent using each specific strategy, $s \in S$, given some observations, O, made online. Given that the parameters of s are known, and assuming a fair deck that deals each hand equally likely, the likelihood of a player using a given strategy can be computed via Bayes rule: $p(s|O) = \frac{p(O|s)p(s)}{p(O)}$. As in explicit Bayesian opponent modeling, the opponent model can be chosen either as the MAP strategy or with importance sampling. Alternatively, an adversarial bandit algorithm such as Exp4 [1] can be used to find the best strategy from the best responses to the portfolio strategies. Implicit modeling with a portfolio of 2010 ACPC competitors is currently the state-of-the-art in exploitive computer poker agents [2]. This paper uses importance sampling over a set of five opponent models for its implicit opponent modeling agent.

[1]Since Leduc poker is a two-player, zero-sum game, all correlated equilibria are also Nash equilibria.

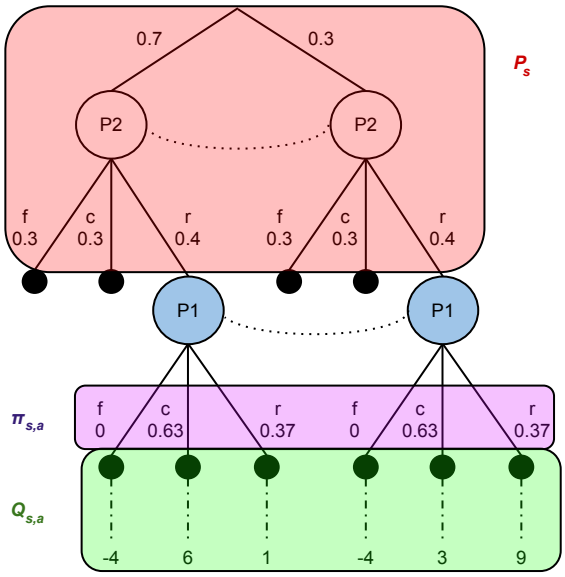

Figure 1: An illustration of how Counterfactual Regret Minimization can be cast as a dynamic programming algorithm and, consequently, transformed into a temporal difference (TD) learning algorithm. The probability distribution over the nodes before a given information set correspond to the transition function, P, for that information set; the value of taking a given action in a specific information set is the Q-value function; and the function mapping an information set to a probability distribution over actions is the policy function π.

3. ALGORITHMS

In this section we present our four novel algorithms for opponent exploitation: TD-CFR, policy bootstrapping, implicit subpolicy modeling, and subpolicy bootstrapping.

3.1 TD-CFR

The *TD-CFR* algorithm transforms CFR from a dynamic programming algorithm to a temporal difference learning method by separating out the transition, value, and policy functions. Figure 1 shows a diagram of where each of these functions is located. TD-CFR, detailed in Algorithms 1-3 is a frequentist, off-policy method that learns an ϵ-regret strategy for a given target opponent. After each hand, the algorithm updates its transition function, P, and value function, Q. Note that TD-CFR assumes all holecards are revealed at the end of the hand, though this could be circumvented by marginalizing the holecard probabilities in a Bayesian approach. Based on the updated P and Q functions, the counterfactual regret is calculated and both the average (off-) policy and the next (on-) policy are updated.

3.2 Policy Bootstrapping

Standard explicit modeling often takes a large number of samples to produce an effective counter-strategy, as it requires maintaining independent prior distributions at every information set in the opponent model. However, once the model has been learned, it can be used to generate a best response or repeatedly sampled to produce variants of the

Algorithm 1 TD-CFR

Input: exploration rate ξ, exploration decay γ^{ξ}, learning rate α, learning rate decay γ^{α}
$\pi_0 \leftarrow$InitializePolicy()
$\pi_{avg} \leftarrow \pi_0$
for all $(s, a) \in S \times A$ **do**
  $Q(s, a) \leftarrow r^{max}$
  $P(s, a, s') \leftarrow$equal probability
  $Regret(s, a) \leftarrow 0$
end for
iteration count, $i \leftarrow 1$
loop
  episode history, $h \leftarrow \{\}$
  reach probability, $reach \leftarrow 1$
  while *playinghand* **do**
    $s \leftarrow$ current information set
    $\pi_i \leftarrow$UpdatePolicy$(Regret, s, reach, \pi_{i-1})$
    $rand \leftarrow$uniform random $[0, 1]$
    if $rand < \xi$ **then**
      $a \leftarrow$randomly selected action
      $h \leftarrow \{(s, a)\}$
    else
      $a \leftarrow$action sampled from π_i
      $h \leftarrow h \cup (s, a)$
    end if
    $reach \leftarrow reach \times \pi_i(s, a)$
    $ev \leftarrow$ExpectedValue(Q,s)
    UpdateRegret$(Regret, s, a, reach, ev, Q)$
    $i \leftarrow i + 1$
  end while
  for all $(s, a) \in h$ **do**
    $P \leftarrow$UpdateP(s, a, Q, α)
    $Q \leftarrow$UpdateQ(s, a, Q, α)
  end for
  $\xi \leftarrow \xi \times \gamma^{\xi}$
  $\alpha \leftarrow \xi \times \gamma^{\alpha}$
end loop

Algorithm 2 UpdatePolicy

Input: total counterfactual regret $Regret$, information set s, reach probability $reach$, current policy π_{i-1}, iteration i
$R^{prev} \leftarrow Regret(s)$
$R^{prev,+} \leftarrow$positive previous regret
$probs \leftarrow$probabilities proportional to $R^{prev,+}$
$\pi_i \leftarrow \pi_{i-1,-s} \cup (s, probs)$
$\pi_{avg} \leftarrow \frac{1}{i}((i-1) \times \pi_{i-1} + reach \times \pi_i)$
return π_i

Algorithm 3 UpdateRegret

Input: total counterfactual regret $Regret$, information set s, reach probability $reach$, selected action a_{chosen}, expected value ev, value function Q
for all a_i in $A(s)$ **do**
  $cfr \leftarrow Q(s, a_i) - ev$
  $Regret(s, a_i) \leftarrow Regret(s, a_i) + cfr$
end for

same type of opponent. Thus, if an explicit model could be efficiently learned, it could potentially be used to generate more robust responses than a simple implicit modeling agent. Obtaining the benefits of a generative opponent model with the quickness of an implicit model is the main motivating benefit behind the *policy bootstrapping* algorithm.

Algorithm 4 Policy Bootstrapping

1: **Input:** portfolio of strategies $\mathcal{S}$, initial strategy s_0
2: $\mathcal{M} \leftarrow$InitializeOpponentModel(s_0)
3: $m \leftarrow$ImportanceSampling$(\mathcal{M})$
4: $s \leftarrow$BestResponse(m)
5: **for all** $s_i \in \mathcal{S}$ **do**
6: $p(s_i) \leftarrow 1$
7: **end for**
8: **loop**
9: **while** *playinghand* **do**
10: $I \leftarrow$current information set
11: $a \leftarrow$ImportanceSampling$(s(I))$
12: take action a
13: $h \leftarrow h \cup$ next opponent action
14: **end while**
15: **if** opponent hand revealed **then**
16: **for all** $s_i \in \mathcal{S}$ **do**
17: $p(s_i) \leftarrow p(s_i) \times p(outcome|s_i)$
18: **end for**
19: **else**
20: **for all** $s_i \in \mathcal{S}$ **do**
21: $x_{marg} \leftarrow 0$
22: **for all** possible opponent holecards hc **do**
23: $x_{marg} \leftarrow x_{marg} + p(outcome|s_i, hc)$
24: **end for**
25: $p(s_i) \leftarrow p(s_i) \times x_{marg}$
26: **end for**
27: **end if**
28: $s_p \leftarrow$ImportanceSampling$(\mathcal{S})$
29: $\mathcal{M} \leftarrow \mathcal{M} + s_p$
30: $m \leftarrow$ImportanceSampling$(\mathcal{M})$
31: $s \leftarrow$BestResponse(m)
32: **end loop**

Algorithm 4 shows the policy bootstrapping algorithm in detail. A portfolio of precomputed opponent models, such as those used for implicit modeling, are given as input to the algorithm. Line 2 initializes an explicit opponent model using a baseline strategy, such as a uniform probability distribution or Nash equilibrium. The algorithm then performs importance sampling at every information set in the opponent model on line 3 to sample a single opponent strategy for exploitation. The best response to the sampled strategy is then played for one hand on lines 9-14. After the hand is complete, lines 15-28 update the pseudo-likelihood estimates for each of the portfolio models and performs importance sampling to choose a best-fit model for the next hand. The best-fit model's strategy is then added to the priors of the explicit opponent model in line 29, updating the beliefs about how the opponent behaves in a more sample efficient manner than direct updates which would only update the information sets along the observed trajectory.

3.3 Implicit Subpolicy Modeling

Rather than modeling a complete policy, either explicitly

or implicitly, it may be more efficient to model components of an opponent's strategy. The key idea is that actions across various information sets are correlated, so when you see an example of an action in one information set, you can update your beliefs about other information sets. As an intuitive example, human players that are observed being aggressive when a specific draw is on the board in Texas Hold'em will likely also be aggressive when other similar draws are on the board, while it may reveal little about how they play in vastly different situations. Such correlations have been hypothesized previously in the literature [14, 15] and modeling opponents based on high-level features that discriminate situations based on the round and action has been used successfully in practice [13].[2] Thus, *implicit subpolicy modeling* is presented as an approach to distinguish between the building blocks of an opponent's strategy. For all of our experiments, the implicit subpolicy modeling agent uses two subpolicies: preflop and flop actions.

3.4 Subpolicy Bootstrapping

Our final algorithm, *subpolicy bootstrapping* combines explicit model bootstrapping with policy decomposition to yield an approach that weaves together an explicit opponent model from a set of subpolicies automatically discovered from a portfolio of complete strategies.

Given a portfolio of strategies and a baseline strategy (e.g., a Nash equilibrium strategy), Algorithm 5 computes a set of subpolicies, where each subpolicy is applicable to some subset of the information sets in the opponent model. Lines 4-7 compute the deviation of the policy from the baseline strategy at information set as a proportion of the baseline strategy. For example, if the baseline strategy at a specific information set has the distribution $(0.1, 0.5, 0.4)$ and the portfolio policy has the distribution $(0.15, 0.4, 0.45)$, then the deviation at this information set is $(0.5, -0.2, 0.125)$. Lines 9-23 then perform a simple clustering of information sets based on *L2*-distance.[3] Lines 24-26 enforce a minimum size constraint on the subpolicies. Finally, on line 33 the algorithm returns the resulting set of all discovered subpolicies.

Once a set of subpolicies has been selected, either automatically or by hand, the subpolicy bootstrapping algorithm proceeds in a manner similar to that of full policy bootstrapping but with increased granularity and some statistical corrections to maintain the correct Bayesian priors. We refer the reader to our implementation for details on the main subpolicy bootstrapping algorithm.

Algorithm 5 Subpolicy Discovery

1: **Input:** portfolio of strategies $\mathcal{S}$, baseline strategy s_b, the minimum deviation δ_{min} from the baseline required to be a member of a subpolicy, the maximum deviation δ_{max} allowed from the centroid of a subpolicy to be considered a member, the minimum subpolicy size $size_{min}$
2: $\Psi \leftarrow \{\}$
3: **for all** $s_i \in \mathcal{S}$ **do**
4: $\quad\mathcal{D} \leftarrow \{\}$
5: $\quad$**for all** information set I in s **do**
6: $\quad\quad\mathcal{D}_I \leftarrow s_i - s_b$
7: $\quad$**end for**
8: $\quad\psi \leftarrow \{\}$
9: $\quad$**while** $|\mathcal{D}| > 0$ **do**
10: $\quad\quad centroid \leftarrow \emptyset$
11: $\quad\quad subpolicy \leftarrow \emptyset$
12: $\quad\quad$**for all** $I, \mathcal{D}_I \in \mathcal{D}$ **do**
13: $\quad\quad\quad$**if** L2distance$(\mathcal{D}_I,(0,0,0))< \delta_{min}$ **then**
14: $\quad\quad\quad\quad$remove I from $\mathcal{D}$
15: $\quad\quad\quad$**else if** $centroid = \emptyset$ **then**
16: $\quad\quad\quad\quad centroid \leftarrow \mathcal{D}_I$
17: $\quad\quad\quad\quad subpolicy \leftarrow \{(I, s_i(I))\}$
18: $\quad\quad\quad\quad$remove I from $\mathcal{D}$
19: $\quad\quad\quad$**else if** L2distance$(\mathcal{D}_I, centroid)< \delta_{max}$ **then**
20: $\quad\quad\quad\quad subpolicy \leftarrow subpolicy \cup \{(I, s_i(I))\}$
21: $\quad\quad\quad\quad$remove I from $\mathcal{D}$
22: $\quad\quad\quad$**end if**
23: $\quad\quad$**end for**
24: $\quad\quad$**if** $|subpolicy| > size_{min}$ **then**
25: $\quad\quad\quad\psi \leftarrow \psi \cup subpolicy$
26: $\quad\quad$**end if**
27: $\quad$**end while**
28: $\quad$**if** $\psi = \emptyset$ **then**
29: $\quad\quad\psi \leftarrow s_i$
30: $\quad$**end if**
31: $\quad\Psi \leftarrow \Psi \cup \psi$
32: **end for**
33: return Ψ

4. EXPERIMENTS

Experiments were performed on all four of our proposed algorithms and the standard approaches from the literature. We next detail our experimental setup and the results for each model in each experiment.

4.1 Setup

To analyze the performance of each model, we conducted a series of experiments in the game of Leduc poker. Leduc

[2]The notion of a subpolicy in this case is related to that of a feature set mapping to states. However, we separate the two concepts here as the distinction is important in subpolicy bootstrapping.

[3]The choice of clustering algorithms is arbitrary; other approaches such as k-means could also be used.

is a two-player poker variant where the deck contains six cards (two jacks, two queens, and two kings), each player is dealt a single hole card, there two rounds of betting, and a single community card is dealt after the first round. In total, Leduc contains 144 information sets for the second player, making this game non-trivial for agent modeling while still being tractable to analye in-depth. For more information on Leduc poker, see [14].

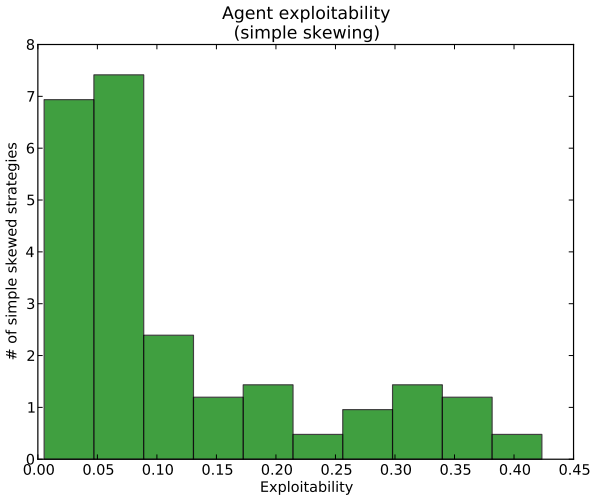

Figure 2: The maximum exploitability of the agents in the simple skewing population.

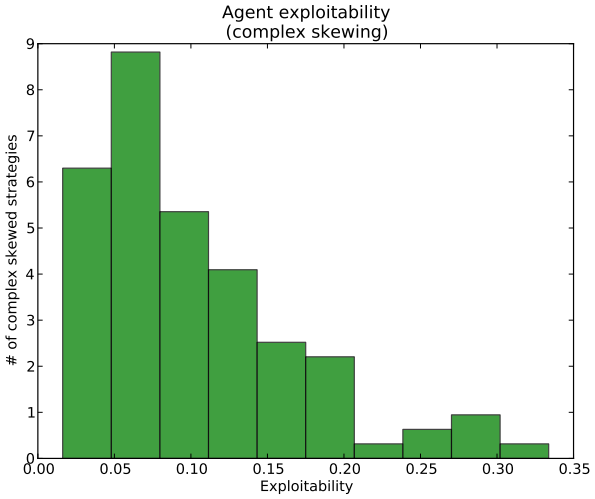

Figure 3: The maximum exploitability of the agents in the complex skewing population.

Experiments were conducted against three types of stationary opponents. First, a population of 100 "simple" agents was generated by generating two random percentage triplets for each agent; the first triplet was then used to skew the preflop actions of a Nash equilibrium strategy and the second was used to skew the flop actions. A population of 100 "complex" agents was then generated in a similar manner, where with equal probability the agent's preflop or flop actions were skewed, with a similar approach used to skew actions based on the holecard of the opponent (J, Q, or K). These two populations of agents aim to capture the systematic bias presumably found in human players that often over- or under- play a given hand or play looser or tighter at different points in the game. Finally, a Nash equilibrium strategy was used as a baseline to examine how the agents perform when trying to model an optimal player. Figures 2 and 3 show the distribution of exploitability values for the simple and complex experiments, respectively, setting an upper bound on how well an agent could perform against them in theory.

In each experiment, the algorithms were tested for 200 matches of 200 hands each, with each algorithm playing against each simple and complex opponent twice. The TD-CFR agent used $\xi = 0.1$ with an exponential ξ-decay factor of 0.99 and a learning rate of $\alpha = 0.05$. For portfolio-based methods, the portfolio consisted of the Nash equilibrium strategy and four skewed strategies, chosen at random from the population at the start of each match. For explicit models, two times the Nash strategy was used as an initial prior. The subpolicy discovery algorithm used the Nash equilibrium strategy as the baseline method, $\delta^b_{min} = 0.05$, $\delta^{sp_i}_{max} = 0.1$, and a minimum subpolicy size of 3.

4.2 Results

Figures 4 and 5 show the results of the five approaches against the simple and complex skewed opponents, respectively. Note that in all cases, the TD-CFR results have been omitted as they performed too poorly to be plotted on the same graph without sacrificing the resolution of the other approaches. We refer the reader to the appendix for the graphs that include the TD-CFR results.

The two implicit modeling approaches appear to perform the best, achieving the highest expected exploitation at nearly every step. Interestingly, the bootstrapped explicit model is able to approach approximately the same level as the implicit models by the end of the run, suggesting that bootstrapping an explicit model may be a viable approach to speeding up generative model learning. Unfortunately, subpolicy bootstrapping fails to perform as well as the other three novel approaches.

Figure 6 presents the results of the six approaches against a Nash equilibrium player. Note that these learning curves are much smoother since each of the 200 trials is against the same opponent, reducing the variance in the evaluation process. Again, we see the two implicit modeling approaches performing strongest, with the bootstrapping methods approaching them in the end.

5. DISCUSSION

The results of the three Leduc poker experiments raise interesting questions regarding the performance of each of the four novel methods relative to the benchmark approaches. We next discuss possible reasons for the performance of each model and speculate about the general potential of each approach.

The TD-CFR model, not shown in the performance figures due to its poor performance, shows little potential for improvement. Typically, we are concerned with one of two situations in poker: computing a strong strategy offline or adapting to exploitable opponents online. In the case of the former, direct methods like vanilla CFR or MC-CFR are

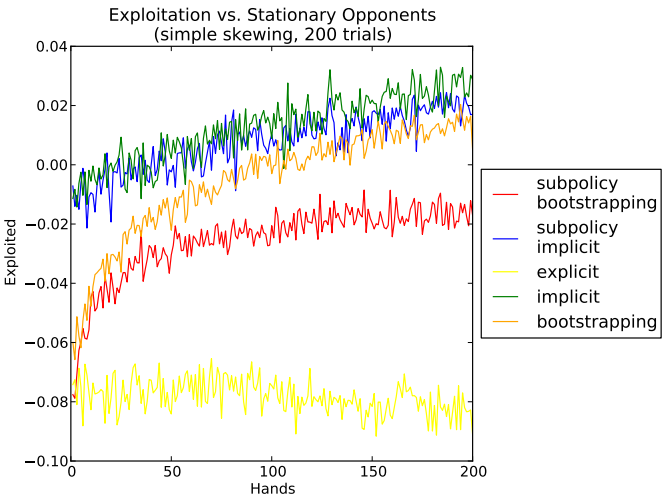

Figure 4: The average performance of each algorithm against the population of exploitable agents derived from a simple strategy skewing methodology. Note that values near each other are not statistically significant due to overlapping confidence intervals.

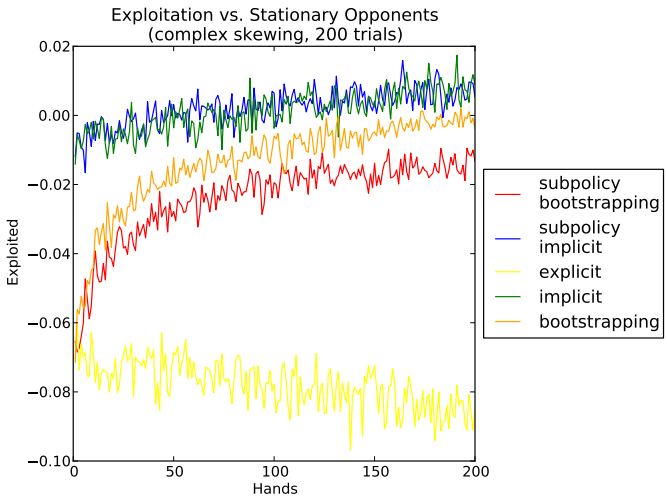

Figure 5: The average performance of each algorithm against the population of exploitable agents derived from a relatively complex strategy skewing methodology. Note that values near each other are not statistically significant due to overlapping confidence intervals.

likely more efficient than the TD-CFR approach. On the other hand, the CFR-inspired approach of TD-CFR makes for slow adaptation that can take hundreds of thousands of games to reach a good policy. Thus, the TD component of TD-CFR makes it ineffective for offline use and the CFR component makes it ineffective for online use, producing a method that unfortunately inherits the worst aspects of both approaches.

The policy bootstrapping results are possibly the most positive of all. Although not as efficient as implicit modeling in this domain, the ability of policy bootstrapping to quickly learn an explicit generative model suggests that it is a promising approach. It may be possible to start from a small portfolio of two or three models, play online against a new opponent, learn an explicit model of their play cheaply (or even profitably), and add the learned model into the portfolio for the next match. Such an approach could prove highly effective in real-world games where gathering a large portfolio directly would be prohibitively expensive.

Unfortunately, both implicit subpolicy modeling and subpolicy bootstrapping failed to outperform their complete policy counterparts. In the case of implicit subpolicy modeling, the performance is virtually identical, indicating that the potential gain for subpolicy modeling in Leduc may be too small to outweigh the increase in model complexity. To examine this possibility, we conducted an experiment with 1k samples of randomly chosen implicit portfolios and opponents, where we measured the maximum exploitation of the opponent using the best response to each portfolio strategy, along with the maximum exploitation of the opponent using every possible subpolicy combination (using preflop and flop subpolicies) of the portfolio.

Figures 7 and 8 show the results of the experiment for the simple and complex skewing populations, respectively. In both cases, an overwhelming majority of the samples produced less than 0.01 bets-per-hand gain compared to the best complete policy response. Additionally, upon inspection of the game tree, only 9 information sets are contained in the preflop betting round as opposed to 135 for the flop round.[4] These results suggest that Leduc may generally be too small for round-based subpolicy partitioning. This may also help explain why subpolicy bootstrapping failed to outperform full policy bootstrapping.

6. FUTURE WORK

The experiments conducted in this paper reveal several opportunities for future research. In this section, we highlight some of the areas that we believe are ripe for future work:

- **Learning opponent models from incomplete information.** Techniques like Data Biased Responses (DBRs) [10] require a large number of well-chosen, perfect-information samples to form strong opponent models that can then be used online. In contrast, an approach like Deviation-Based Best Resonses (DBBRs) [5] can potentially model opponents through incomplete information observations, but requires a startup phase of T hands to exploit an opponent. Ideally an exploitive agent would develop models that are both immediately exploitable and feasible to gather in practice.

- **Increasing sample efficiency.** It seems intuitive that most real-world opponents' exploitable actions are correlated in a systematic way; this intuition has been noted by other researchers [15] as well. Increasing the information derived from opponent observations, whether from subpolicy analysis or some other

[4]The flop betting round is also twice as expensive as the preflop round.

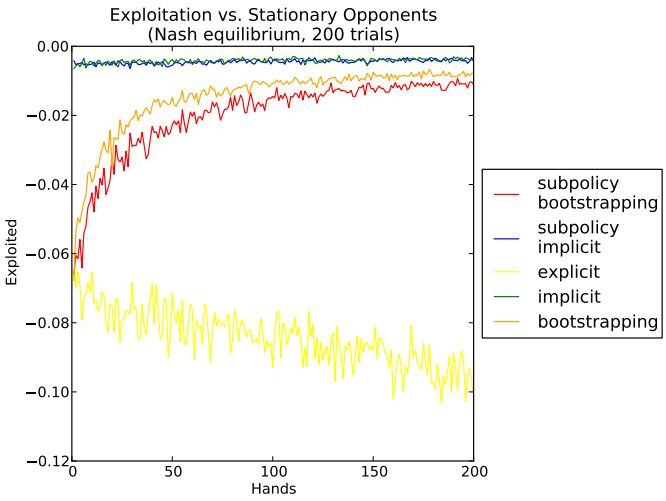

Figure 6: The average performance of each algorithm against a Nash equilibrium agent.

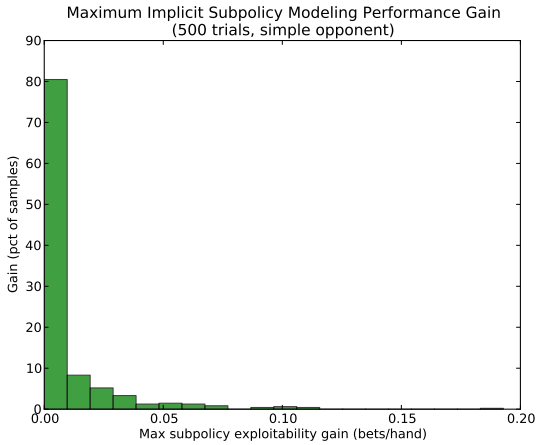

Figure 7: Results of 500 random trials calculating the maximum exploitation gain from using an implicit subpolicy approach instead of full policy implicit modeling. The vast majority of trials result in less than 0.01 bet/hand potential gain, indicating a possible reason for the failure of the implicit subpolicy modeling algorithm.

approach, is critical to the success of future exploitive agent research.

- **Subpolicy discovery.** Uncovering which subregions of opponent policies may contain the same systematic bias holds a large amount of potential. The straightforward approach to subpolicy discovery through better feature engineering is a pragmatic approach, but a more general approach would be more powerful and be applicable to a wider class of games.

- **Incremental best response calculation.** When maintaining an explicit opponent model where updates are made to only a portion of the model at each point, calculating a full best response may be unnecessary. In games such as Texas Hold'em, the full best response may be too computationally expensive to compute after every hand. An approach that can update only the parts of the model that are effected at each turn could vastly speed up the algorithm, with important practical implications.

- **Modeling non-stationary opponents.** The vast majority of exploitive play in poker has assumed a stationary opponent, likely due to the presumed drastic increase in difficulty of opponent modeling. Fortunately, while this may be the case for game theoretic non-stationary agents, humans appear to use relatively simple models that are weighted heavily towards recent observations [4]. Broadening the class of opponents to include such simple agents may hit the sweet spot of analytic tractability and real-world applicability.

- **Deceptively teaching opponents.** All exploitive poker research to date has ignored the potential for teaching opponents an exploitable model. It seems reasonable that situations would present themselves, particularly against humans or agents using "foresight-free" models [4], where an agent could cheaply teach an opponent a policy that is exploitable for a much larger amount.

The above list has highlighted only some of the opportunities in the area of online opponent modeling. Given the small amount of research in the area, there may be numerous additional fruitful avenues of research.

7. CONCLUSIONS

This paper has presented four novel approaches to online opponent exploitation in poker. The TD-CFR method was proposed as a temporal difference learning variant of the CFR dynamic programming algorithm, and was found to perform poorly in all experiments. Three alternative, Bayesian approaches were then explored: policy bootstrapping, implicit subpolicy modeling, and subpolicy bootstrapping. None of these methods outperformed the standard implicit modeling benchmark, however policy bootstrapping was able to achieve significant speedups over direct explicit modeling. An in-depth study was conducted to determine the reason for the failure of the subpolicy methods and we concluded that Leduc may be too small of a test domain for such subpolicies to show any benefit, even when systematic biases are introduced into the opponents. We concluded with a list of potential areas for future work in opponent exploitation, in addition to exploring our four approaches on more challenging domains.

8. REFERENCES

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

[2] N. Bard, M. Johanson, N. Burch, and M. Bowling. Online implicit agent modelling. *Autonomous Agents and Multiagent Systems (AAMAS'13)*, 2013.

[3] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*,

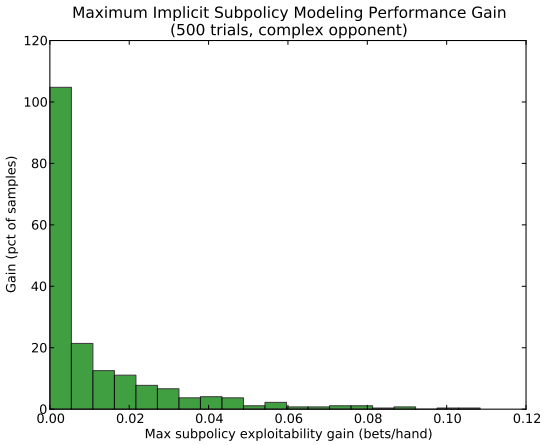

Figure 8: Results of 500 random trials calculating the maximum exploitation gain from using an implicit subpolicy approach instead of full policy implicit modeling. The vast majority of trials result in less than 0.01 bet/hand potential gain, further supporting the hypothesis that Leduc does not benefit from subpolicy modeling.

134(1):201–240, 2002.

[4] I. Erev and A. E. Roth. Multi-agent learning and the descriptive value of simple models. *Artificial intelligence*, 171(7):423–428, 2007.

[5] S. Ganzfried and T. Sandholm. Game theory-based opponent modeling in large imperfect-information games. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 533–540. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[6] A. Gilpin, S. Hoda, J. Pena, and T. Sandholm. Gradient-based algorithms for finding nash equilibria in extensive form games. In *Internet and Network Economics*, pages 57–69. Springer, 2007.

[7] A. Gilpin and T. Sandholm. A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1007. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[8] S. Hart and A. Mas-Colell. A general class of adaptive strategies. *Journal of Economic Theory*, 98(1):26–54, 2001.

[9] M. Johanson, N. Bard, M. Lanctot, R. Gibson, and M. Bowling. Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, pages 837–846. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[10] M. Johanson and M. Bowling. Data biased robust counter strategies. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, pages 264–271, 2009.

[11] M. Johanson, M. Zinkevich, and M. Bowling. Computing robust counter-strategies. *Advances in neural information processing systems (NIPS'07)*, 20:721–728, 2007.

[12] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling. Monte carlo sampling for regret minimization in extensive games. *Advances in Neural Information Processing Systems (NIPS'09)*, 22:1078–1086, 2009.

[13] M. Ponsen, K. Tuyls, M. Kaisers, and J. Ramon. An evolutionary game-theoretic analysis of poker strategies. *Entertainment Computing*, 1(1):39–45, 2009.

[14] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and D. C. Rayner. Bayes' bluff: Opponent modelling in poker. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pages 550–558, 2005.

[15] F. Southey, B. Hoehn, and R. C. Holte. Effective short-term opponent exploitation in simplified poker. *Machine learning*, 74(2):159–189, 2009.

[16] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[17] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. *Advances in neural information processing systems (NIPS'08)*, 20:1729–1736, 2008.

APPENDIX

A. TD-CFR RESULTS

The graphs from section 4.2 omitted the TD-CFR results so as to enable better presentation of the other methods. For completeness, we include the full graphs here.

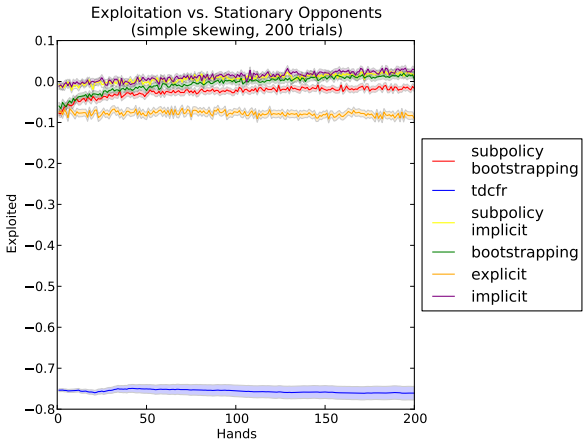

Figure 9: The average performance of each algorithm against the population of exploitable agents derived from a simple strategy skewing methodology.

B. CONFIDENCE INTERVALS

Due to the large number of time series, confidence intervals were left off of the graphs in section 4.2. For completeness, we include them here.

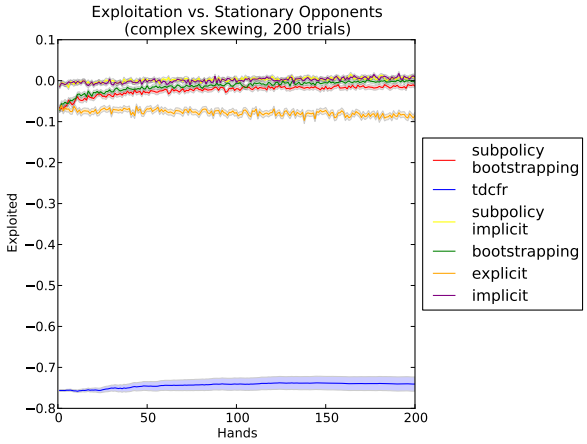

Figure 10: The average performance of each algorithm against the population of exploitable agents derived from a relatively complex strategy skewing methodology.

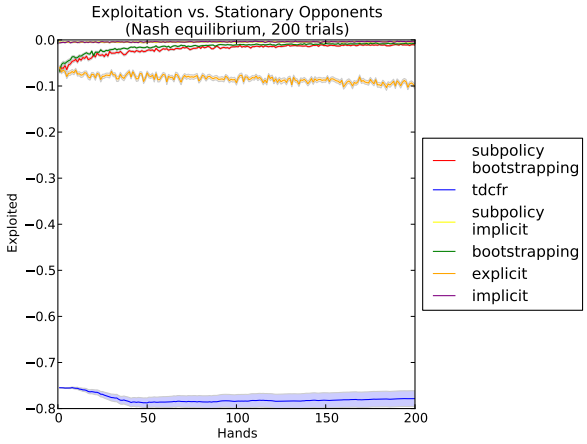

Figure 11: The average performance of each algorithm against a Nash equilibrium agent.

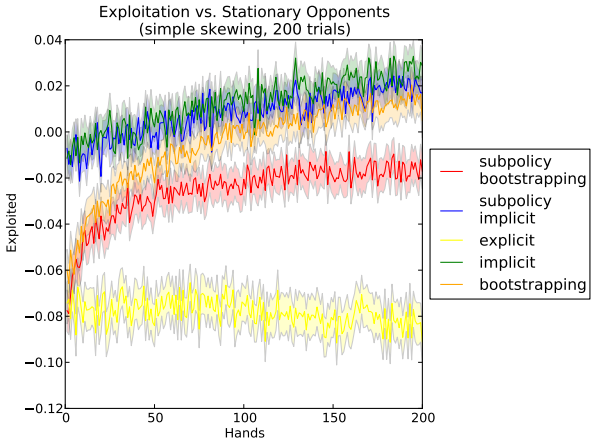

Figure 12: The average performance of each algorithm against the population of exploitable agents derived from a simple strategy skewing methodology.

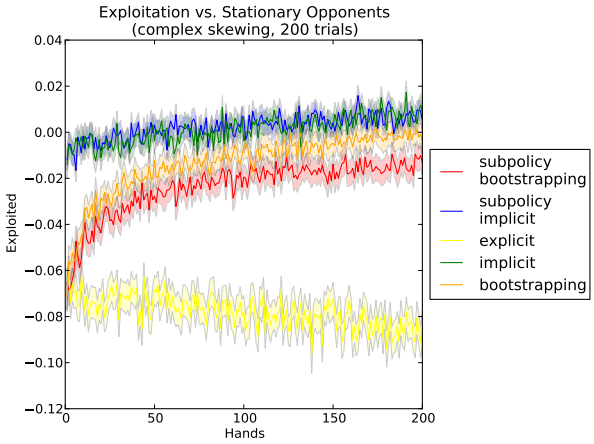

Figure 13: The average performance of each algorithm against the population of exploitable agents derived from a relatively complex strategy skewing methodology.

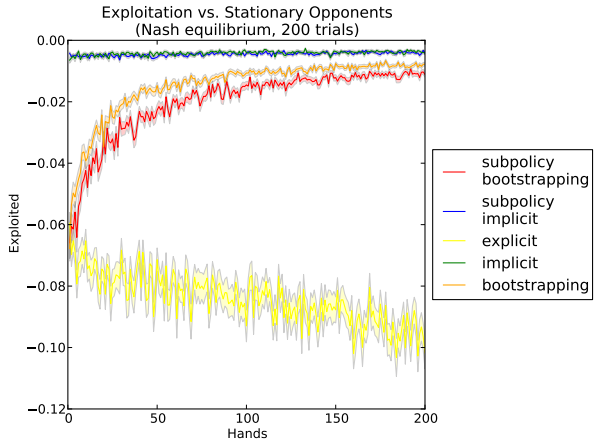

Figure 14: The average performance of each algorithm against a Nash equilibrium agent.