Development of a Heads-Up Autonomous Poker-Playing Agent

Trevor Cappallo
cappallo@gmail.com

May 15, 2013

Contents

1 Texas Hold'em in brief **3**

2 Notable prior work **4**
2.1 Counterfactual Regret Minimization 4
2.1.1 ϵ-Nash equilibria 5
2.1.2 Drawbacks 5
2.2 VexBot 6

3 Current machine learning strategies **6**
3.1 Opponent modelling 7
3.2 Bucketing 7
3.3 Dealing with Variance 8

4 My approach **8**
4.1 Abstraction 9
4.2 Weighted Monte Carlo runoffs 9
4.3 Balancing with CFR 10

5 Results **10**
5.1 Against chump-bots 10
5.2 Against strong bots 13
5.3 Against humans 13

6 Conclusion **13**
6.1 Further study 13

1 Texas Hold'em in brief

Hold'em has been the dominant form of poker for the past decade or two, and is the topic of most of the current research in the field. I've tried to condense it to the relevant essentials; here is a concise overview of basic play.

- Requires 2–10 players.
- Up to four rounds of betting:
  - 2 cards are dealt to each player privately ("hole cards"), after which there is a round of betting ("preflop").
  - 3 cards are dealt face up in the center as shared community cards, after which there is a second round of betting ("the flop").
  - Another card is dealt face up in the center, a third round of betting ("the turn").
  - A final fifth card is dealt in the center, and a final round of betting ("the river").
- Each player's hand is formed by taking the best five-card standard poker hand they can make out of the seven cards available to them (their two hole cards plus the five shared community cards).
- Each player has at least one chance to act per betting round:
  - If there is a bet to them, they can stay in by either calling the bet, or raising it; otherwise they fold and forfeit their chance at the pot.
  - If there is no bet to them, they can either bet or "check," which is essentially the same as betting nothing and staying in the hand for free.
  - A player wins the hand (and all money in the pot) if she either has the best hand on the river, or if everyone else in the hand has folded (in which case she does not need to show her cards, a critical detail).
- The two most popular betting structures in Hold'em are "no-limit" or "fixed-limit."

- No-limit allows any player to bet as much as they want when it is their turn to act, up to the total amount of chips they have available; it is generally considered a tougher, more dynamic game.
- Fixed-limit standardizes the bet and raise amounts, so e.g. a player must bet in $1 increments; due to the much smaller state space, this form has been a more attractive target for AI research (as well as novice poker players) thus far.

2 Notable prior work

As the popularity of poker exploded in the early 2000s, academic interest in it as a vehicle for AI and machine learning techniques appeared as well. The vast majority of it has been coming from the Computer Poker Research Group out of the University of Alberta, who have been releasing papers steadily for a decade and consistently put together programs which rank at the top in computer poker tournaments. Of necessity, most of the ideas I discuss in this paper originated there.

2.1 Counterfactual Regret Minimization

The majority of the top competitive poker-playing agents incorporate a technique called Conterfactual Regret Minimization (CFR). "Regret" is closely related to the concept of utility value. The **overall average regret** of a player i is

$$R_i^T = \frac{1}{T} \max_{\sigma_i^* \in \sum_i} \sum_{t=1}^{T} (u_i(\sigma_i^*, \sigma_{-i}^t) - u_i(\sigma^t)). \tag{1}$$

This equation represents player i playing T games (hands) of poker, using the strategy $\sigma_i^t \in \Sigma_i$, where a strategy denotes an action (or more typically a probability distribution to select an action) at each node t in the extended form state space. u_i is that player's utility function, where the superscript * denotes an improved strategy, and the subscript $_{-i}$ denotes all strategies other than those selected by i. Essentially, the overall regret is a measure of the discrepancy between how a player actually played, and the strategy yielding the maximum overall average utility.

2.1.1 ϵ-Nash equilibria

Zinkevich et al. [5] proved that minimizing **immediate counterfactual regret** node-by-node bounds overall average regret, and as a consequence, minimizing the immediate regret necessarily lowers overall regret. By repeatedly traversing the game tree and minimizing regret, you approach a Nash equilibrium.

In a zero-sum game, a Nash equilibrium may be informally defined as a set of two or more strategies where any deviation from the strategy by one player has a strictly worse outcome than their previous strategy; in other words, no player can benefit by unilaterally altering his play. An ϵ-Nash equilibrium describes strategies that approach a true Nash equilibrium within a factor of ϵ, so that the most that a player utilizing one of these strategies can possibly be exploited by is ϵ per game.

The general implementation of CFR involves two agents stepping through an arbitrary number of games against each other, closely resembling training through self-play. ϵ will asymptotically approach 0 as the number of iterations increases. To turn this into a single viable strategy, one generates an average strategy profile as an average of the two players' strategies.

2.1.2 Drawbacks

CFR is an extremely powerful tool in that it approaches "perfect" play—that is, it is guaranteed not to lose by more than the arbitrarily small ϵ, and therefore agents employing it are nearly impossible to beat. However, Nash equilibrium strategies tend to be conservative in that they capitalize little on opponents' mistakes. It is perhaps not dissimilar to an expert chess player who is sadistically determined to draw. Since most substantial profit in live poker is dominated by the exploitation of mistakes of weak players by strong players, I find that CFR is not a complete solution to poker, but rather a useful foundation.

It is also worth noting that the success of a CFR approach depends heavily on the level and sophistication of the abstraction one chooses to employ. With a sufficiently compact abstraction[1], we can reduce ϵ to 0 and play a perfect game. However, if your abstraction is too small or simple-minded, there is no guarantee that it will translate well back to the extensive-form game. In fact, there has been recent research on **abstraction pathologies**

[1] As of this writing, the largest tractable abstraction consists of some 10^{13} states.

indicating that there are several classes of problems that tend to emerge in these situations[4].

2.2 VexBot

VexBot[1] was designed by Darse Billings et al. in 2006 as an attempt to create a bot more capable of playing in real world situations. Specifically, they realized that one doesn't typically get to play tens of thousands of hands against an opponent, as in typical computer poker tournaments: a truly competitive agent must be able to learn and adapt over the course of hundreds or dozens of hands.

However, this necessarily means using a much coarser abstraction than those employed by the more defensive CFR agents. Having relatively few data points to work with, a large number of states would be next to useless. Instead, they leaned hard in the other direction—the primary indicator VexBot relies on is nothing more than the number of bets and raises made by both players, summed together, a mere 9 states. However, VexBot has proven to be an extremely powerful heads-up player, adapting to and defeating almost any opponent after a couple of hundred hands.

3 Current machine learning strategies

The foundation of many of the top computer players is to develop an ϵ-Nash equilibrium strategy via CFR or one of several improved CFR-derived algorithms which have been recently developed[2]. While such a strategy is guaranteed not to lose by much, it necessarily cannot win by much either. It is unable to exploit adversaries' poor play, and it is such exploitation that drives the real profit in poker.

It is therefore crucial to vary the software's play to adapt to its opponent, presumably by modeling said opponent in some way and using that model to capitalize on apparent weaknesses in play. From what I've ascertained, it seems like the best programs stitch together a number of approaches to form a complete poker-playing package. Neural nets are currently popular for this purpose, but there are a number of alternatives in the literature. The key is perhaps not to rely on any single strategy, but rather to work in tandem with more specialized modules or heuristics.

3.1 Opponent modelling

This opponent modeling is made problematic by the partial-information nature of Hold'em. In "heads-up" play—a one-on-one match, which is the subject of most extant literature due to the more managable state space—a large majority of hands are not played to completion, meaning that the poker agent gains only indirect indicators of how its adversary is playing. Conclusions can still be drawn from this—for example, if the adversary plays every single hand aggressively, it is safe to assume that he is often betting strongly with mediocre and poor hands—but it is a slower and more complicated process than a setup in which all of the adversary's hands are viewable after the fact. This transparency is common in the literature where opponents are often other poker bots with readily accessible models of play whose cards can be examined at will, but is virtually unknown in real-life play.

3.2 Bucketing

Because the state space of a full Hold'em game is far too large to be tractable (2-player fixed-limit has some 10^{18} game states[3], no-limit many more), this must be mitigated by introducing one ore more simplifications. For my project, I am currently planning on implementing a heads-up no-limit agent. Furthermore, the best poker-playing agents all use some form of abstraction to represent a full game of poker, the most popular of which is "bucketing." During each round of play, an attempt is made to partition hands with similar properties into the same bucket. A typical agent will use about a half dozen of these buckets during each stage of the game, which provides a managable branching factor.

It seems that the most straightforward and effective selection criterion for bucket partitioning is to analyze the expected outcome of the agent's hand against a uniformly random adversary hand, simulating as-yet-undealt community cards as necessary. In the simplest case, you could create two buckets, one for every above-average-strength hand, and one for every below-average-strength hand. It has also been established that performance is significantly improved when the expected value of the square of the hand strength is considered, rather than the raw hand strength, as this emphasizes the value of drawing hands which have a solid chance to improve to a strong hand in a later round. I plan to adopt this approach in my own program.

3.3 Dealing with Variance

Hold'em is an extremely "noisy" game, being possessed of a high degree of chance in any one hand. While this is one of the aspects of it that attracts researchers—reasoning under uncertainty and partial information has obvious broader applications—it also makes poker a difficult nut to crack. **Variance** is often used informally in poker contexts to refer to this random element, and it makes it very difficult to determine which of two players is actually winning. If they are sufficiently closely matched, it can take millions of hands to get a strong enough signal to determine the stronger player with a 95% certainty[1].

To help ameliorate this problem, researchers have developed several techniques to cut down on this requirement. One technique is playing **duplicate poker**, where two computer players will play a certain number of hands, and then have any memory gained reset, and play through the same hands on opposite sides. This has been shown to reduce the standard deviation in winnings by about a factor of four, a significant savings.

Another approach is **DIVAT analysis**, DIVAT being a tool developed by the CRPG to conduct post-mortem analyses of games. In essence, it consists of running a very straightforward baseline player through those same hands; on the ones that were clearly favorable for one player, the DIVAT agent will also tend to do well, and will discount that "luck" when measuring the performance of the players. DIVAT also reduces standard deviation by a factor of three or four, and can work in tandem with duplicate poker.

4 My approach

I am attempting to build a hybrid between a CFR agent and an aggressively adaptive player heavily inspired by VexBot. The idea is to implement CFR, which requires an *a priori* extended self-training session of days or weeks. My bot will use CFR as its default fallback strategy; when it is unsure what to do, or does not yet have a good read on its opponent, it will stick to playing by CFR, ensuring that it loses little or nothing while it is gathering data on its opponent. On top of that base, I will build a more exploitative agent that can take charge when it seems advantageous.

4.1 Abstraction

For opponent modelling, I am beginning with a similar model to VexBot, tracking the statistical likelihood of my opponent's action at any given state based on his behavior in previous similar states. As with VexBot, "similar" is a bit generous, as the model must consist of a relatively limited number of unique states in order to derive any kind of statistical inference from it.

More specifically, I am tracking the relative frequency of each kind of action $a \in A$ that occurs after t actions have taken place during that round. When the data are initialized, a small chance for every action is the default so that no branch of the state space is prematurely cordoned off, but those values are quickly dominated by actual online data gathered as play progresses. As an example, if it is the seventh action during a given game (a 'game' and a 'hand' are synonomous in this context), and so far we have seen the opponent fold twice there, call once, and raise five times, we go ahead and assign relative likelihoods to those actions of 2/8, 1/8, and 5/8 respectively.

4.2 Weighted Monte Carlo runoffs

The very basic analysis described above is then combined with a number of Monte Carlo runoff simulations. In each iteration, the game is played from the current point through to the end, with random community cards being dealt each time if any are still to come. The opponent's hand is also randomly dealt.

Rather than being truly random Monte Carlo iterations, instead the choice of which branch to take is dictated by any statistical data we have gathered about each node visited. For example, if an iteration were to encounter the node described above, it would go down the 'raise' path 62.5% of the time, the 'call' path 12.5% of the time, and would fold (which is a terminal action) 25% of the time. Although this approach is moderately more computationally expensive than a full-random Monte Carlo sim, I suspect that it is worth it in that the majority of our limited processor time and memory will be allocated toward analyzing areas of the state space that are more likely to occur. With my bot being written in C and utilizing the popular open-source library **poker-eval** to handle much of the low-level card representation and manipulation, my program is able to play out tens of thousands of games per second.

4.3 Balancing with CFR

Once CFR is implemented, I will use a simple weighting system to determine whether to use a player-model action suggestion or CFR's suggestion. If the player model module predicts relatively high profit compared to CFR's analysis, or if the player model has a sufficient amount of data to make it more reliable at a given node, the bot will give proportionally more weight to selecting the player model's action. When it is a close call, they will be selected with roughly even odds so as to decrease predictability.

5 Results

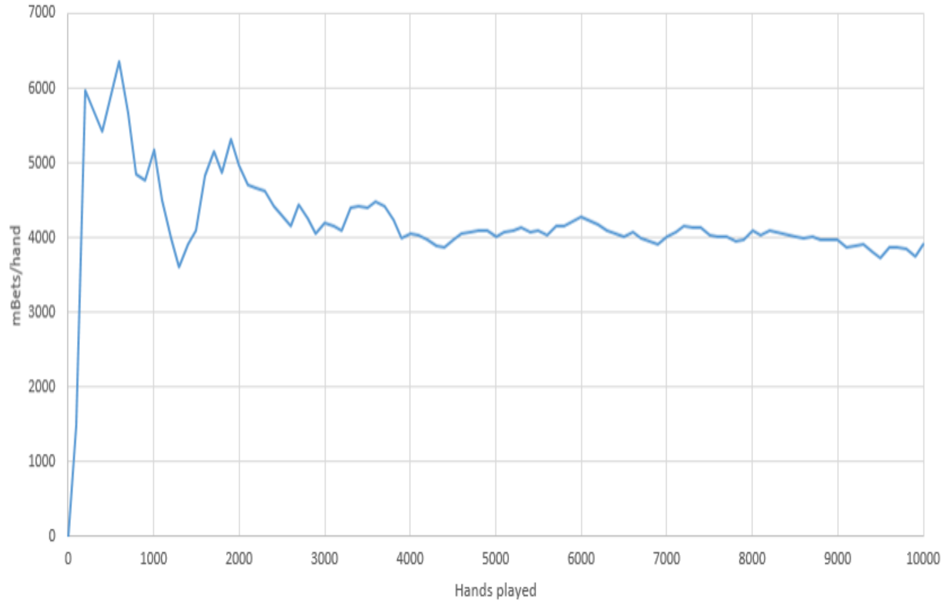
Overall, I have a reasonably solid program developed. It supports arbitrary chump-bot opponents, several types of output, live "vs-human" play, a speed/depth setting, any number of persistent opponent- and self-models, and CSV-format result logging.

Although I have CFR implemented, I have not yet finished debugging it; it is still losing at a slight rate to the chump bots. Additionally, it is a challenge to structure a sufficiently sophisticated abstraction while staying within resource boundaries; applying several gradations of raising and/or half a dozen buckets causes the stored game tree to rapidly climb into gigabytes of RAM use. However, the bet pattern analyzer portion is working, and I have been able to test it against a couple of "chump bots" and make some optimizations.

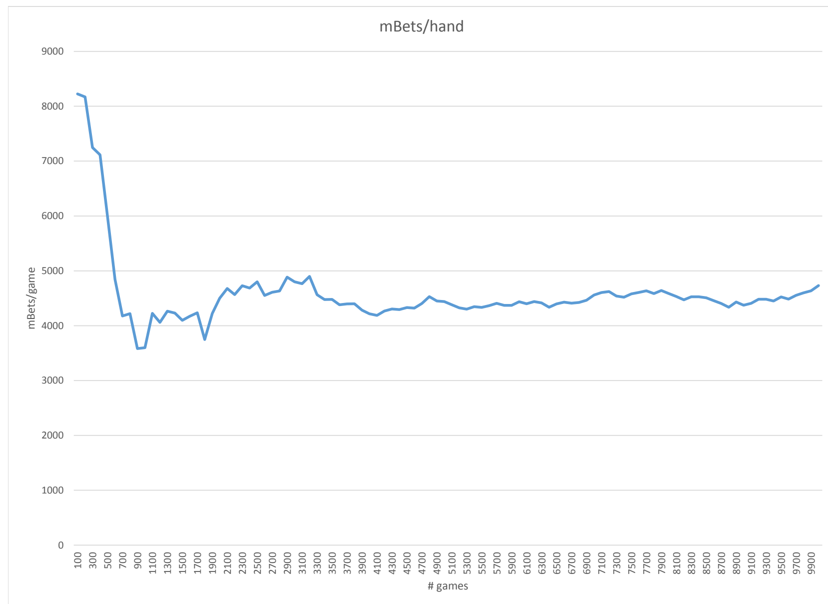
5.1 Against chump-bots

There are fundamentally only two interested so-called "chump bots", which are agents that are trivial to program and thus make for good early sparring partners and measuring sticks. One is a player who always calls, and the other is a player who always raises. You can do mixes of these, and throw in folds, but it turns out there's not much more to be learned from that.

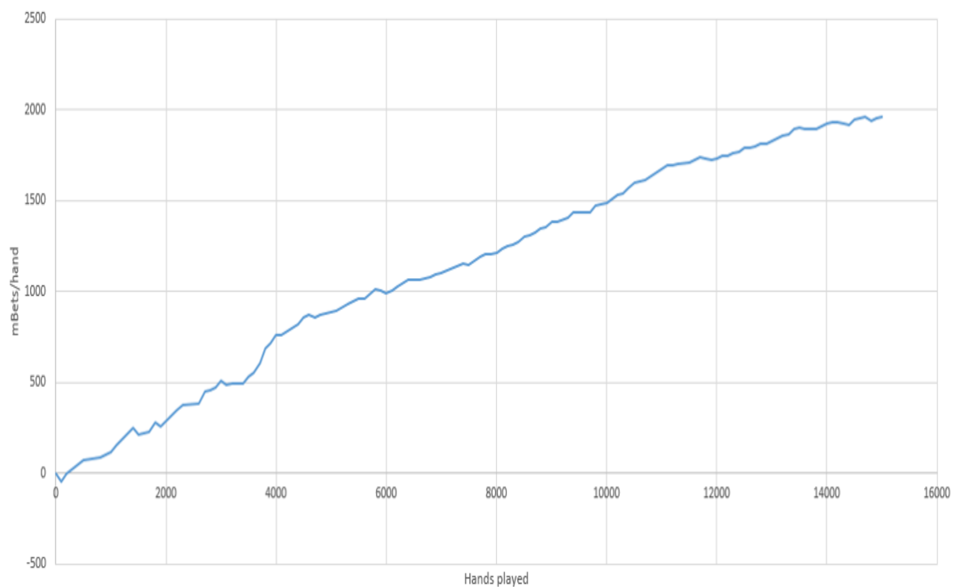
My earlier prototype of the betting analyzer which I had ready for our class presentation was much slower to pick up on the opponent's strategies. Here we have before-and-after graphs of play against both the calling and raising chump.

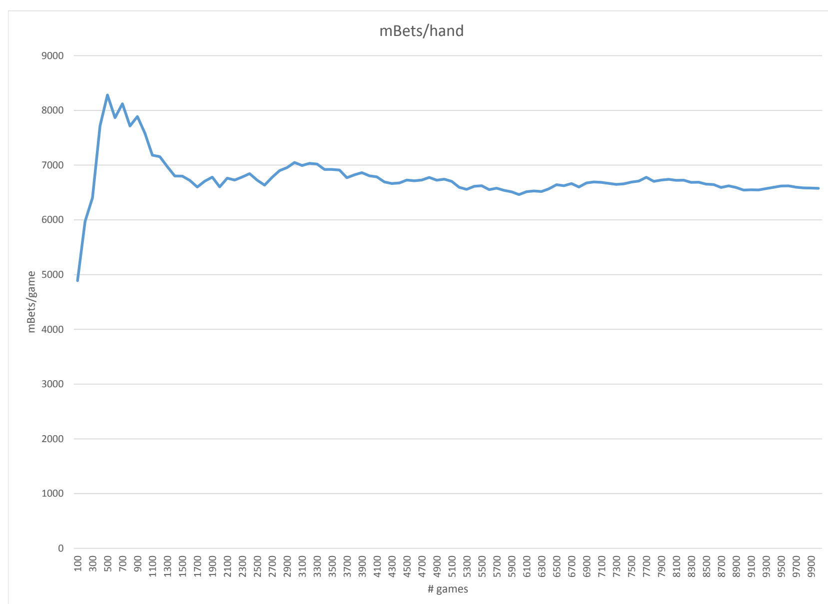


Win rate against RaiseBot
mBets/hand
Hands played
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000



mBets/hand
mBets/game
games
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Win rate against CallBot
mBets/hand
Hands played


mBets/hand
mBets/game
games

5.2 Against strong bots

Unfortunately, people tend to be very protective of more capable poker bots, as they have an obvious financial value. I'd wanted to test my program against some better competition, but was unable to find anything that was open-source or remotely compatible with my project. It remains on my radar, however.

5.3 Against humans

This is tough to test since humans aren't generally prepared to sit down and play 50,000 hands of poker in a row against a computer. However, I stepped up to the challenge and played two 50-hand matches against my betting-analysis bot. I was pleasantly surprised when it won the first match at 2800 mBets ahead. The second match was an exact tie.

Although the sample size here is so small as to render it statistically meaningless, I have many years of poker experience and could tell that it was playing substantially stronger than a novice or a random chump bot. It also appeared to be learning already by the end of our hundred games that, for example, my preflop raises tend to signify strong hands and I saw it's fold rate in that situation increase. When I have time, I'd like to play a thousand hands or so against it, as that seems to be the point at which it leveled out against the chump bots.

6 Conclusion

Although I was disappointed that I was unable to complete my CFR component in time, I suspect it is very close to operational and I plan to continue work on it over the coming days.

The simplistic betting-analysis agent, however, completely surprised me in its apparent skill while utilizing such a comparatively unsophisticated algorithm. From that alone, I consider this project a success inasmuch as I set out to create a viable heads-up no-limit player.

6.1 Further study

There are a number of areas which beg further experimentation and improvement.

- With a few modifications, my bot could be made to play limit poker instead of no-limit. This would be very useful for comparison purposes as there is no shortage of literature giving hard values for limit bot performance.

- Instead of having the betting-analysis agent start from a blank slate every time it faces a new opponent, I would like to have it create a composite default profile based on average play across all opponents to date.

- I believe the betting-analysis agent could be further improved through a handful of other straightforward indicators; for example, I suspect percent of one's total chips wagered during a hand corresponds strongly—and possibly inversely, depending on the opponent—to the true strength of their hand.

- Find a few other human testers for the bot to see whether my experience was singular.

- Once CFR has the kinks worked out, move on to parallelize the code for a substantial speed increase. The same could likely be done with the betting-analysis agent.

References

[1] Darse Billings, Aaron Davidson, Terence Schauenberg, Neil Burch, Michael Bowling, Robert Holte, Jonathan Schaeffer, and Duane Szafron. Game-tree search with adaptation in stochastic imperfect-information games. In *Computers and Games*, pages 21–34. Springer, 2006.

[2] Richard Gibson, Marc Lanctot, Neil Burch, Duane Szafron, and Michael Bowling. Generalized sampling and variance in counterfactual regret minimization. In *Twenty-Sixth Conference on Artificial Intelligence (AAAI)*, pages 1355–1361, 2012.

[3] Denis Richard Papp. Dealing with imperfect information in poker. Master's thesis, University of Alberta, 1998.

[4] Kevin Waugh, David Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathologies in extensive games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 781–788. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[5] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. Technical report, University of Alberta, 2007.