Jason Li, Jenny Li, Jerry Li, Jaeyoon Song (Calling Station)

1/31/23

Pokerbots Final Strategy Report

After training a successful GTO NLHE poker bot with MCCFR last year, River of Blood Hold'em proved to be a difficult variant to adapt to. With the game tree growing exponentially every street, we experimented with parallelization, data compression, and sub-game solving. We constantly ran into memory, time, and storage issues; ultimately, our final bot was quite similar to last year's with a few important modifications.

1. Hand Abstraction

We abstracted Heads-up No Limit Hold-em (NLHE) by mapping similar information sets to single abstract information sets, which reduced the game to a size feasibly solvable by MCCFR. Our hand abstraction method was the same as last year; we grouped similar hands into "buckets." During the preflop round, suit-isomorphic hands were treated as identical, producing 169 buckets. To bucket later streets, we partitioned the 169 preflop hands into eight distinct clusters with unique characteristics, generated each possible hand + board combo, and calculated each hand's equity against the eight opponent ranges. Using the outputted eight-dimensional vectors, we ran the K-means++ algorithm to group the hand + board combos into 256 clusters on each street.

2. Action Abstraction

We reduced the action space by limiting each player to seven possible actions at each node: check, fold, call, 50% pot size raise, 100% pot size raise, 200% pot size raise, and all-in. These raise sizes were obviously more realistic than last year's (75%, 150%, and 300%); however, the smaller sizes and the increase of the starting stack from 200 to 400 meant more possible infosets. We addressed the bot size limitation by storing “purified” actions (see section 4) and the efficiency limitation by parallelizing MCCFR and extending the training time (see section 5).

Like last year, we capped the number of consecutive raises at five and implemented the property of imperfect recall. Each infoset was defined by the history of actions (encoded by k, f, c, x, y, z, a in the order listed above) and the active player’s bucket (encoded by P, F, T, R for pre-flop, flop, turn, and river, followed by the bucket number). We did not include the run. We tried training on an abstract game where the number of streets was capped at seven, but it failed due to the bot size restriction and our suit-isomorphic buckets (which did not work for a variant based on red/black cards). Based on weekly tournaments, neglecting the run during MCCFR still produced strong pre-run actions.

Our abstractions resulted in around 76 million infosets, a significant increase compared to last year’s five million possible infosets.

3. MCCFR

This year, we were able to parallelize external-sampling MCCFR; each thread traversed the game tree concurrently and updated the same hashmap in memory. Only one thread updated strategies (every one thousand iterations) and handled I/O, saving the hashmap to disk periodically. Again, we implemented linear CFR to discount past iterations and regret-based pruning to temporarily prune negative-regret actions and avoid unnecessary traversals of subtrees. We started discounting and pruning much later than last year (after 200 million and 50 million iterations vs. after 20 million iterations) to allow strategies to stabilize. Similarly, we decreased our minimum threshold for regrets from -110,000 to -3,100,000.

Our final bot was trained on a 96-core machine on AWS, which achieved 5 billion iterations over 11 days. This was a vast improvement over last year's bot, which was trained on a Thinkpad X13 for 1.2 billion iterations over three days.

4. Post-Processing & "Manual" Play

After obtaining a strong strategy from MCCFR for the abstract game, it still needed to be translated into the actual game for tournament play. The opponent may make "off-tree" actions; namely, using bet sizes which do not correspond to the ones in our abstraction. As with last year, we used the pseudo-harmonic mapping function to map opponent bets to bets in our abstraction.

Furthermore, given a probability distribution over several actions at each infoset, there are several ways in which to choose the action to take. Following our previous bot and the top bots in literature, we performed the following steps: first, if the fold probability is greater than a certain threshold, we always fold. Otherwise, we group all bet action probabilities into a "meta"-bet action. Then, we choose the action between fold, check/call, and bet of highest probability. If "bet" is chosen, we select the bet size with highest probability. This year, we stored the "purified" action for three different fold thresholds, rather than just one.

Since our blueprint strategy is only used up to and on the river, we needed a different strategy on the run. For this, we put together a very straightforward strategy: we computed our equity against the opponent preflop range, and raised/called with good hands and folded with bad ones. This was passable on the run. We also realized that we needed additional equity computation on the river instead of just relying on the blueprint if the last card is red, since we would fold away many of our drawing hands which still had substantial equity. So, we computed the expected hand strength distribution – to distinguish between drawing hands and made ones – and played slightly differently for each. All of this "manual" play is undoubtedly extremely exploitable, but we were planning to replace it with real-time search (next section) which did not come to fruition.

Finally, as with past years, we check-fold when we've ensured the game is won, and also never fold if this guarantees the opponent a win (instead jamming all-in).

5. Real-Time Search

Our major goal this year was to implement real-time search/subgame solving (RTS), but unfortunately the scrimmage server proved to be too limited in compute resources for it to be feasible. However, since a large portion of our effort was directed to this goal, we dedicate a section to it nonetheless.

We chose to attempt RTS on every street in the run. This is necessary because it is not possible to store a blueprint strategy for even the first street of the run within the size constraints. We also did not attempt RTS before the run – like Pluribus does – because we needed the size of the subgame to be as small as possible (to be tractable with the time and compute limitations). We implemented unsafe subgame solving, which in practice often produces low-exploitability strategies despite lacking theoretical guarantees. To do so, we assumed that both players played by our blueprint strategy up to the run. For every possible hand, we maintained the probability that the opponent would have taken the actions they did with that specific hand, and similarly for ourselves. Hence, when our bot enters the run, it has a probability distribution over all 1,326 possible hands for both the opponent and for itself. Then, we created a new subgame for the specific game situation we're in, with a root chance node that samples a hand for the opponent and for ourselves, and finally we solved the subgame with external-sampling MCCFR – just as with the blueprint. To be tractable, we assumed the hand goes to showdown on the current street, regardless of suit.

Unfortunately, as aforementioned, this is probably ~10x too slow to work effectively on the scrim server, though it is actually close to feasible on our own machines. Therefore, our final bot omitted this section entirely.