Texas Hold'em Poker Bot

Martina Kollárová
kollarova@mail.muni.cz

1 Introduction

This work creates a Poker bot that is able to compete with other bots in the RoboPoker[1] competition. The Texas Hold'em[2] variant was chosen, because it is the most popular and has both hidden (cards in the hand of the players) and shared information (cards on the table). It is also the most commonly used variant in AI research [1].

1.1 Texas Hold'em Poker Rules

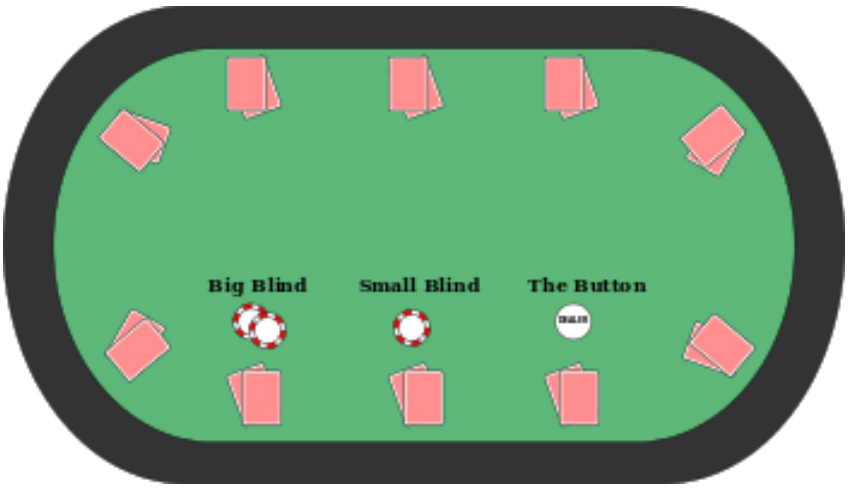

Figure 1: Playing table with initial hidden cards and forced bets.

Figure 1 shows how the game starts. In plays between real players (i.e. face to face), the button marks the player who is going to deal the cards (shuffle them), but it also sets the position of the players with the small and big blind. It is usually moved around the table counter-clock wise, so that players are chosen for the blinds equally. The big and small blinds are *forced bets* that are used to prevent the scenario where all players would commonly fold (give up) before anything was played. The small blind is half the amount of the big blind.

Each of the players has two *hole cards* which only he can see. Going clock-wise, the players choose whether to:

call put enough money on the table to match the highest bet

check special case of call when no money needs to be added to continue (a zero-cost action)

[1] `http://robopoker.org`
[2] `https://en.wikipedia.org/wiki/Texas_hold_'em`

fold give up, lose all money he already bet

raise/bet add extra money to the pot

The *pot* is the money on the table that will be given to the winner of the game. To continue, the player has to have as much money in the pot as the highest bet so far, otherwise he has to fold. For the game to continue to the next round, each of the players who want to stay in the game have to have equal bets in the pot. Even if nobody raises in the first round, the players have to put in at least the big blind in to stay in the game.

The beginning of the game, when no shared (visible cards to all) are on the table, is called *preflop*. When the betting for that round is finished, the dealer puts three shared cards on the table, and the same is repeated—this round is called the *flop*. The betting continues same as before, and afterwards the dealer reveals a fourth card—the *turn*, and after another round of bets he will add the last fifth card, which is called the *river*.

If only one player remains, he has automatically won. But if there are more players left at the end (called showdown), they show their cards and compare their value, according to Figure 2. The rules about special cases (when two players have a similarly ranked card) can be complicated,[3] but this is already handled in RoboPoker.

Figure 2: Poker hand ranking—the higher the better.

[3] `https://en.wikipedia.org/wiki/Texas_hold_'em`

2 Problem Analysis

2.1 Existing Implementations

Bots are usually forbidden in online games, since even though they are worse than the professional players, they can play decently against the average players. They are not limited by sleep or fatigue, and can play multiple games simultaneously, which are considered unfair advantages.

The existing bots are usually not open-source, since they are created to make money (and spend a vast amount of effort going past things like CAPTCHA and pretending to be humans so that the servers don't recognize them as bots, since they are usually not allowed). There are frameworks and tools available for it, but the strategy is usually not published.

2.1.1 OpenHoldemBot

The OpenHoldemBot[4] is a framework for creating Poker bots that contains tools for creating a bot for online Poker games with humans. It mostly deals with screen scrapping, analyzing the game play and functions that are not necessary for us, since we don't need to be able to work with the user interface of online games and the rest is either provided by RoboPoker's SDK or more compact external libraries (like python-pokereval).

2.1.2 Annual Computer Poker Competition

An alternative competition to RoboPoker is the Annual Computer Poker Competition[5] (ACPC), which also provides an XML protocol trough which players can compete. It is well known and attracts both AI researchers and hobbyists, but it couldn't be chosen for this project, because it is only run annually in June.

3 Implementation

3.1 RoboPoker

The RoboPoker competition defines a REST API[6] trough HTTP POST with which the bots can communicate. This makes it possible to implement the bot in any kind of programming language. A request looks as follows:

```
name=bot01&pocket=AH+8C&actions=fold%0Acall%0Araise
&state=%3C%3Fxml+version%3D%221.0%22+%3F.....
```

The bot has to just parse the input, which contains the state of the game as seen from the player in XML (the *state* item), make a decision based on them and return on of the possible actions, for example *raise*. The *name* field allows the bot to choose a strategy based on it, so that multiple types of bots can be hosted on the same URL.

RoboPoker also provides and SDK—it allows to run the bot against other test bots locally, without having to add it to the competition on the website. Therefore, it is not necessary to implement the game engine that would execute the game and decide on the winner.

[4] `https://code.google.com/p/openholdembot/`
[5] `http://www.computerpokercompetition.org`
[6] `http://robopoker.org/about/api/`

3.2 Hand Strength

The hand strength computation is done by an external Python module *pokereval*.[7] It contains pre-computed probabilities that a certain hand, in combination with the cards on the board, will win against one opponent.[8] A similar algorithm is also described in the work by Castillo [2].

Originally, I expected that the algorithm used to create the *pokereval* library didn't count with the possibility that the community cards might be well suited for my bot, but even better for the opponent. For example, if there are four cards of the same suit on the table and I don't have a card of that suit (let's say I have *three of a kind*), it is very likely that the opponent will have a card of that suit and win with the *flush*. I have designed and mostly implemented a rollout simulator, which would try to compute this by randomly sampling a few thousand possibilities as to what the opponent could have in his hand and seeing if I would win against him. However, I have afterwards found out that the algorithm already counts with this and my simulator isn't necessary.

3.3 Effective Hand Strength And Pot Odds

Pot odds are defined [4] as:

$$C = \frac{C}{C+P},$$

where C is the amount of money that is needed to call or raise and P is the amount of money in the pot (i.e. on the table). The number is in the range from 0 to 1 and shows whether it's "worth it" to continue—a small number means the cost of the call is small, compared to the amount that can be won; a number closer to one means that you have to put in a lot of money to win the pot and you should continue only when the chances of winning seem high. For example, if there are already \$100 in the game and we need to add \$10 to call (or we can add \$10 to raise the bet), the pot odds are 0.09.

Since the more opponents are there in the game, the more probable it is that somebody will have a better hand, the hand strength needs to be de-vaulated depending on the number of players, which is currently done as

$$\text{hand\_strength}^n,$$

where n is the number of opponents (hand strength is a number ranging from 0 to 1). This is called *effective hand strength*. In other works [2] it also uses the probabilities that the hand gets worse or better, but in this case this is already calculated into the hand strength by the pokereval library.

Originally, it was also using the number of followers (players who still have the chance to raise after the bot's turn) and the number of raises in the round to de-valuate the hand strength, similar to the equation used in Findler's work [3], but it was showing worse results and was discontinued.

The result is that the bot calls or raises when the pot odds are smaller than the effective hand strength:

$$\frac{C}{C+P} \leq \text{hand\_strength}^n$$

If the above is true, it will call or raise. Therefore if the hand is not very good, but the pot odds are good (only a little money has to be added to win a lot of money), the bot will call or raise (which one actually depends on a hand strength threshold). For example, a hand strength of 0.6 (meaning that the player should win 60% of the time against a single opponent) seems good, but against 4 opponents it becomes 0.1296. However, if the player only has to add \$10 to call and there are \$100 on the table, the pot odds are 0.09 which makes it a reasonable risk and the bot should continue.

[7] `https://pypi.python.org/pypi/pokereval/`

[8] `http://www.suffecool.net/poker/evaluator.html`

3.4 Probability Triples

The bot uses the probability triples similarly as the work by Castillo [2]. With it, the main decision making is made by the probability triple generation function and the rest can be abstracted away, which made it possibly to drastically simplify the decision logic of the bots and remove code duplication. It also makes randomization of results simple.

A probability triple (PT) is an ordered list of three values, $PT = (f, c, r)$ such that $f + c + r = 1.0$. Each value represents the likelihood that the next action is a fold (f), a call (c), or a raise (r), respectively.

The function that chooses the action based on the triple is shared among the different bot strategy implementations, so some details (e.g. some actions not being possible in the current state) can be handled by it. If the raise action was chosen, but isn't possible, it will try to call. If calling is not possible, it will try to check. If checking is not allowed, it will fold. Thanks to this, none of the bots will fold when they can check, which is an action with no cost.

3.5 Strategies

The basic strategies are as follows:

random	choose to fold/call/raise with equal probabilities (if "check" is available and "fold" was chosen, check)
simple	always call if we can, never raise
threshold	raise if hand strength is above some threshold, call if it is lower but higher than another threshold

The strategies that use the effective hand strength and pot odds described in Section 3.3 are referred to as *smart*. The variations use different thresholds for deciding on whether to call or raise. The main distinction is between the *aggressive/passive* strategies (raising often or little) and the *loose/tight* strategies (playing many hands or only the best ones) [5]. To implement the *loose* and *tight* variations, there is another threshold constant for each strategy—if the bot is in pre-flop and the hand strength is higher than that threshold, he will play the card.

agressive-loose	raise even if cards are not so good; in preflop play cards that are not too great
agressive-tight	raise even if cards are not so good; in preflop only play cards that are very good
passive-loose	raise only if cards are very good; in preflop play cards that are not too great
passive-tight	raise only if cards are very good; in preflop only play cards that are very good
average smart	has threshold values in the middle of the aggressive/passive and loose/tight
randomized	same as as "average smart", but choose a different action than normally, with some small probability

4 Usage

To install the dependencies for the bot, use:

```
$ pip install -u -e .
```

The bot can be executed by running

```
$ python wsgi.py
```

which will cause it to listen at the URL `http://localhost:8080`. It requires that the HTTP POST parameters described in Section 3.1 are specified. The *name* parameter will decide what the strategy is, and can be one of *random, simple, threshold, agressive-loose, agressive-tight, passive-loose, passive-tight, smart, randomized-smart*, as described in the previous section. Thanks to this parameter, the single URL can act serve as multiple bots at the same time.

RoboPoker's SDK provides tools to test the bot locally, which can be done by editing the file `hand_players.list` and adding a line as:

```
3   smart             200     http        http://localhost:8080
4   agressive-tight   200     http        http://poker-mkollaro.rhcloud.com
```

where the first column is the sitting position (has to be unique in that file), second is the name of the bot (strategy), third is the amount of money the player brings in at the beginning, then the protocol and URL. The first line in this example points to our locally running bot, the second is the publicly available copy of it. Then you can generate a random deck of cards with `create.sh` and play it with a humanly readable output as follows:

```
./create.sh | ./play-human.sh
```

An example of such output is in Listing 1. The first section shows what are the hidden cards of the players (this is meant for debugging, it's usually not accessible). The *PREFLOP* section shows what are the initial forced bets (small blind and big blind) and what were the actions of the bots (ordered by time). The section *POTS* shows how much money is on the table at the end of the round. The sections *FLOP, TURN* and *RIVER* work in a similar way as PREFLOP. In *SHOWDOWN* it shows the winner and his winning combination name and cards—the first number of a card is its rank, the second is the suit (H is hearts, S is spades, C is clubs, D is diamonds). In this case, the bot with the aggressive and tight strategy won with a pair of cards (two cards with the rank 7).

5 Evaluation and Testing

After the project was started, it was found out that the RoboPoker competition was discontinued. The creators were contacted and it was started again, but most of the enlisted bots seem to be non-functional. My bot was first in the rankings for many weeks. After I created a post[9] about it, two additional players joined, and some of the creators of the older bots were contacted. However, even after that, there seem to be only four four bots that are able to play on a basic level, but one of them is currently beating mine, as shown in Figure 3. The bot used in the competition in the average *smart* bot. Even though other version of the bot showed better results locally, the website didn't allow me to change it (it is still a beta version, adding another bot would require that I create another account and deploy the bot service on another URL).

The strategies were also compared among each other, as shown in Figure 4. It shows the money that was left to the bots after 3000 rounds, when each of them started with $200 (the starting money is included in the chart, so a bot with less than $200 left lost money). The results are as expected—the bots with the smart strategy are performing much better than the others.

[9] `http://reddit.com/r/programming`

```
DEAL
   simple       3H TD
   agressive-tight 7C 2S
   smart        2D TC
PREFLOP
   simple[190] small_blind[10]
   agressive-tight[180] big_blind[20]
   smart[200] fold[0]
   simple[180] call[20]
   agressive-tight[180] check[20]
POTS [40]
FLOP          5H3CKS
   simple[180] check[0]
   agressive-tight[180] check[0]
POTS [40]
TURN          7D
   simple[180] check[0]
   agressive-tight[140] bet[40]
   simple[140] call[40]
POTS [120]
RIVER         4H
   simple[140] check[0]
   agressive-tight[100] bet[40]
   simple[100] call[40]
POTS [200]
SHOWDOWN
agressive-tight wins 200 with Pair [5H, 3C, KS, 7D, 4H, 7C, 2S]
```

Listing 1: Example output of a game

The Figure 5 shows a similar comparison, but of the variations of the smart strategy, in addition to the *threshold* bot. Again, it is the result after 3000 rounds of poker, with $200 as the starting money. It shows that the smart bot with the aggressive and tight strategy is winning against this group of players. This is not surprising, because it is often recommended [5] to play only a few best hands (unless the player is the small/big blind) and raise often. The randomized strategy is not very useful against deterministic bots as shown here—its purpose is to diminish the predictability of the bot, so it cannot be easily categorized by an opponent modeller or human.

Additionally, the project contains a basic set of unit tests, which use nosetests,[10] mainly for the parsing of the state represented in XML.

[10] `https://nose.readthedocs.org`

RoboPoker βeta About API SDK Materials Rating My bots My tables mkollaro@gmail.com (41420.00$)

Rating

2014-05-30 — 2014-05-31

#		Account	Rating	Prev.	Best
1		aqu	81934	1	1
2		**mkollaro** it's you	64113	2	2
3		Gaer	12580	3	1
4		vradrus	2480	4	4
280	▲	vbo	-275	293	4
281	▲	botQ.ru	-8775	290	279
282	▼	anonymous	-8940	279	278
283	▲	anonymous	-9340	285	278
284		lenny	-9360	284	276
285	▲	anonymous	-9550	287	283
286	▲	MisterX	-9740	291	282
287	▲	anonymous	-9950	288	277
288	▼	anonymous	-10290	283	277
289	▼	anonymous	-10450	280	279
290	▼	anonymous	-10490	282	280

Figure 3: Results of RoboPoker competition

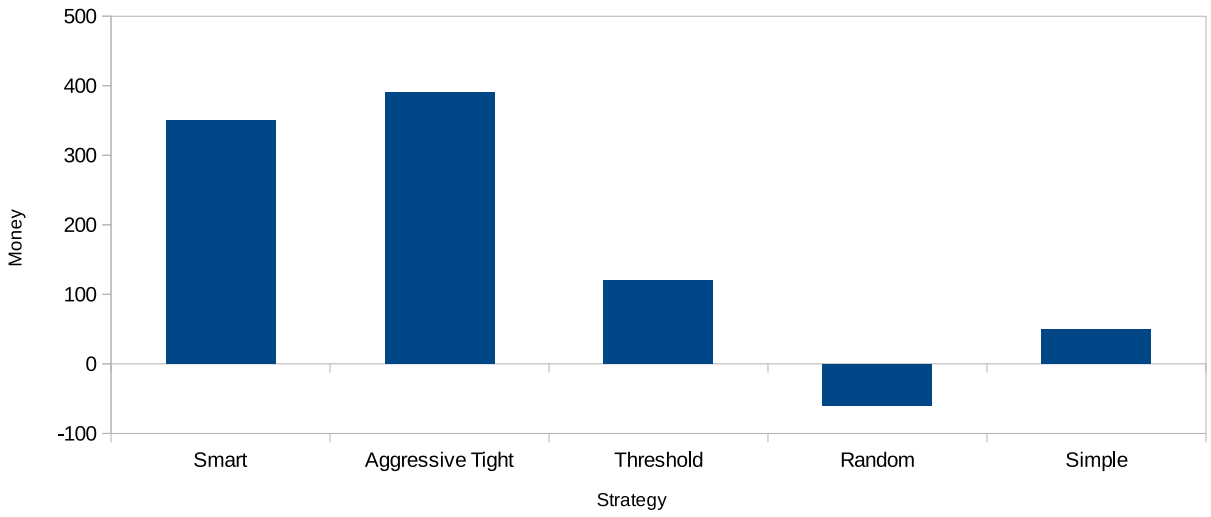

Figure 4: Comparison of bots

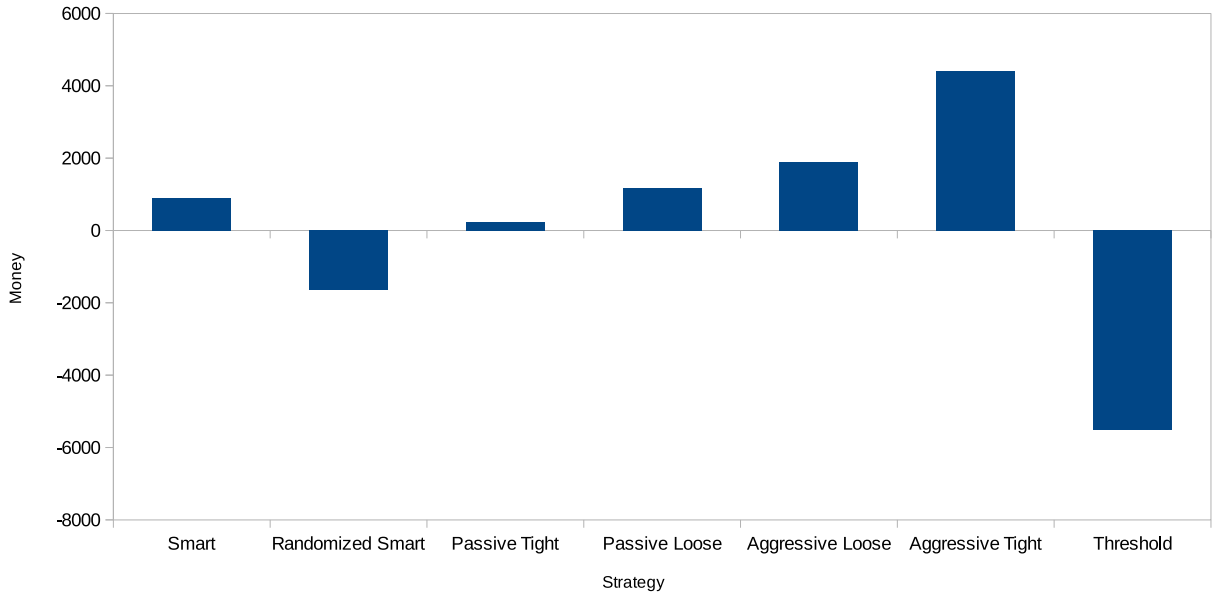

Figure 5: Comparison of Smart bot variants and Threshold bot

6 Conclusion

The bot was created in Python and published[11] under the Apache 2 licence, so that people can easily fork it and write their own strategy, without having to implement details like state parsing. Anyone can play with the bot trough the HTTP API even without participating in the RoboPoker competition, since it is publicly available as a service.[12]

The bot has ranked as second in the RoboPoker competition with the average "smart" strategy and provides multiple variations of it for comparison, of which the best was found to be the one using the aggressive and tight strategy.

The bot is currently only rule based, with optionally random decisions. Since the RoboPoker's API doesn't support result collection and fixing it would be a difficult task, the opponent modelling wasn't implemented. In addition to that, the bots in the competition don't seem to be very smart. The research on opponent modelling could be done as a Bachelor's or even Master's thesis.

[11] `https://github.com/karinqe/poker`
[12] `http://poker-mkollaro.rhcloud.com/`

References

[1] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Poker as a testbed for ai research. In RobertE. Mercer and Eric Neufeld, editors, *Advances in Artificial Intelligence*, volume 1418 of *Lecture Notes in Computer Science*, pages 228–238. Springer Berlin Heidelberg, 1998.

[2] Maria de Lourdes Peña Castillo. Probabilities and simulations in poker. Master's thesis, University of Alberta, 1999.

[3] Nicholas V. Findler. Studies in machine cognition using the game of poker. *Commun. ACM*, 20(4):230–245, April 1977.

[4] Kevin B. Korb, Ann E. Nicholson, and Nathalie Jitnah. Bayesian poker. In *In Uncertainty in Artificial Intelligence*, pages 343–350. Morgan Kaufman, 1999.

[5] D. Sklansky. *The Theory of Poker*. Two Plus Two Pub., 1999.