The Art and Practice of Poker Simulation

Scott Ostler
Professor Davis
6.871
5/12/2005

Table of Contents

Description of Task

Task Overview

The task chosen was to construct a Knowledge-Based System for the playing of No Limits Texas Hold-'Em, a popular variant of Poker. Specifically, the program will advise whether to, given a set of two cards at the beginning of the hand and knowledge of the table betting, fold, call, or raise. The problem consists of three very different sub-problems: assessment of the relative strength of the hand, assessment of the table position and the bets made by opponents, and an intelligent combination of those two assessments to form a recommendation of betting strategy to make.

Texas Hold-'Em Rules

Texas Hold-'Em is a variant of Poker played with a normal 52 card deck. Standard Poker rules apply for determining hand rank. In most casinos, play proceeds in the following manner:

1. All players are dealt two cards, face down. They may be looked at. The player to the right of the dealer must put in an amount of money to the pot known as the "Little Blind" amount. The player to the right of him (that is, two positions to the right of the dealer) must put in the "Big Blind" amount to the pot.
2. Now begins a "round" of Poker. All players after those two must upon looking at their hand, decide to call (match the round's bet), make a raise (increase the round's bet), or fold (put in no more money and forfeit the round). This is done by circling around the table, going to each player continually until all players who haven't forfeit have put in an equal amount.
3. After a round of Poker, cards are turned over in the middle of the table. These form a communal pool which all players use to form hands, along with a player's two face down cards which are used only by the player they were dealt to. Cards

are turned over in quantities of three, then one, then one, with interleaved rounds of betting. There is a final round of betting. Finally, hands are compared, with the highest hand winning the pot.

Different variations constrain the betting. In the Limit variation, there is a limit of how much any player can raise. In No-Limit, no such restriction exists, and the maximum bet that can be made is to go "all-in". There are also often limits put upon the number of raises that can be made in a given round.

My program specifically advises on the initial round of betting described in Step 2, given the Limit variation. That is, it only deals with the round wherein each player has their two cards in front of them, with no communal cards. It handles the case in which it the player has already made a bet this round.

Example:

```
(ask [Bet Dan ?x] #'print-answer-with-certainty)

What is Dan's First Card: King

What is Dan's Second Card: King

What is Dan's # of Callers: 3

What is Dan's # of Raisers: 2

Has Dan Put in Money: Yes

Output: Bet
```

Problem Solving Paradigm

Rule Based Reasoning

I choose to use a rule based system to encapsulate my knowledge. The first and most trivial advantage of using a rule based system was my familiarity with the approach and the tools, due to Problem Set 2. More deeply, I believe the domain of Poker in general lends itself to this approach, and is the most appropriate medium by which the Expert and I can converse.

My reasoning is as follows. The nature of Poker inherently lends itself to difficulty in determining the 'correct' action. After all, even given the outcome to a given hand, we may in no way conclude the proper play for that hand. Phil Hellmuff, a famous world champion, is known to say upon defeat that he played flawless poker and was beaten by luck. Although to an extent its probably sour grapes, his overblown point bears a kernel of truth. Rules give structure in a way that Examples, probably the other salient reasoning system, avoid. With a rule, we abstractly define what Poker should be. With rules, we could find ourselves having to answer difficult situational questions. Rules, being general and sweeping, suffer from no such problems. The flipside is that with a Case-drive system, were a given hand routinely messed up by our system, we could easily attack the relevant case. In a complex rule system where effects propagate, in no way can we easily mimic the built-in granularity of a case-based reasoning system. This is an acceptable trade-off.

Knowledge Structure

Knowledge Specification

The knowledge the system requires to make intelligent decisions is an understanding of the values of various poker hands, as well as the impact that the table position will have upon a round. In other words, before the program offers a recommendation, the factors are combined in an arithmetic statement, such that if the hand is sufficiently good or bad enough, the table position is irrelevant. Note that unlike a more probabilistic approach, the program classifies hands into tiers, for both convenience and for a very important underlying reason. When opponents choose to play hands, they generally do so only on hands they suspect they can win. For that reason, a pair of nines is effectively about as good as a pair of jacks, because it will lose to all high pairs. In a table with 10 people, it's likely that there will be such a pair when the cards come down. For this reason, most classes of hands are in general equal to others within that class. The exception is the highest classes, the AK is far, far better than an AQ, and the reason for that is that both hands are going to be popular to play, yet because one 'trumps' the other in the vast majority of situations, the AQ is a much weaker hand than initially suspected, and the AK is a far stronger hand.

What it specifically knows about poker hands:

- The relative rankings of all pairs, from high to low
- The importance of having a straight draw
- The importance of having a flush draw (two cards of the same suit)
- The importance of having high cards

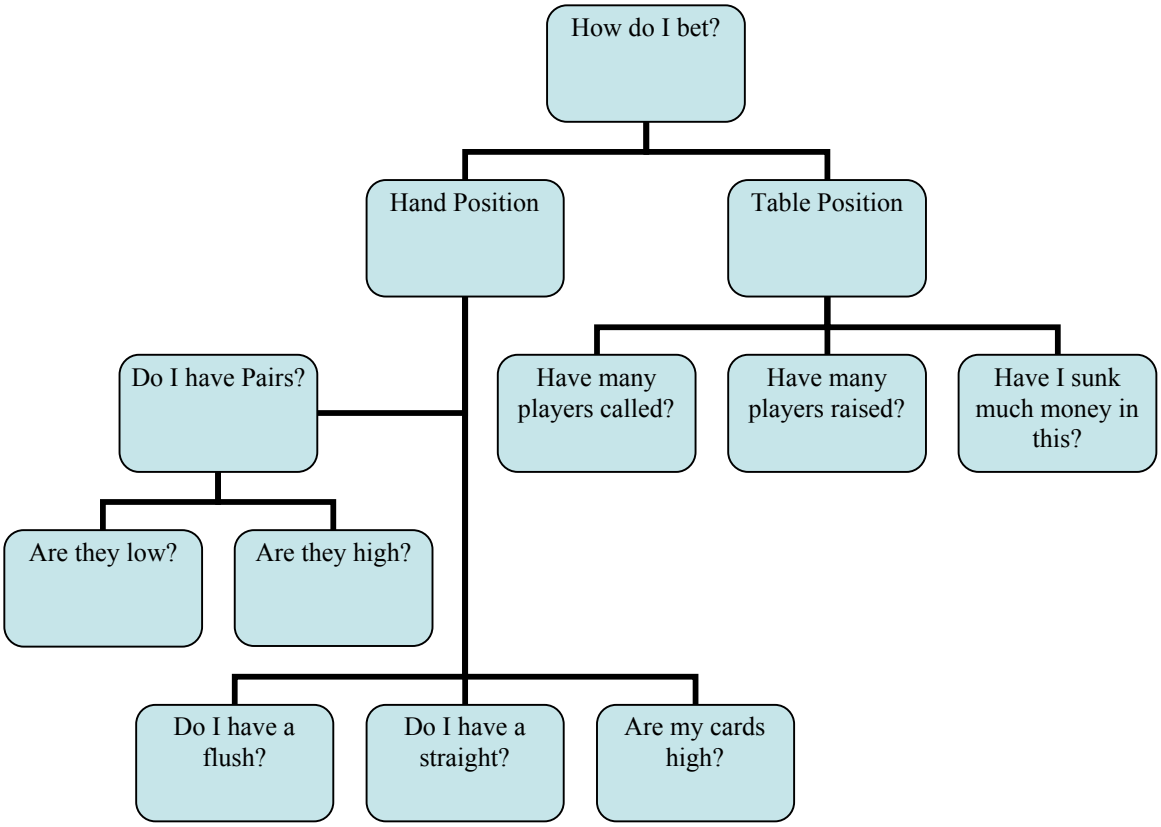
However, previous betting is often critical in giving correct advisement. The program knows:

- The dangers implied when players call
- The dangers implied when players raise
- The difference between when you're already invested in the pot and when you have no stake in the pot.

To be more explicit about the last bullet point, if you have significant money in the pot and players call you, then you should aggressively protect your money. However, in the case that players then raise you, its important to not further increase losses by pursuing a lost pot.

Finally, the program knows how to weight all these factors together and produce an output recommendation of Fold, Call, or Raise.

Conceptual Program Flow Chart



This forms a conceptual outline of the intended flow of the program. However, dozens of "helper" rules exist to convert input from one format to another, circumvent specific limitations I ran into with Joshua, etc.

Example Rules

Two fairly representative rules are as follows:

- (defrule straight (:forward :certainty 1.0)
  if [and [card1-number ?who ?x]
  [card2-number ?who ?y]

```
                (or (= (- ?x ?y) 1)
                    (= (- ?y ?x) 1))]
            then [has-straight ?who yes])
```

- ```
  (defrule pair-of-jacks (:backward)
          if [and [has-pairs ?who yes]
                  [card1 ?who JACK]]
          then [hand-strength ?who 60])
  ```

The first represents a syntactical rule, the other a semantically rule. The first serves to identify whether or not a given hand is a straight draw, that is, whether the two cars are adjacent to each other (Ace and King, for example). We operate on the numerical representation of the cards, as opposed to the named representation. Simply, it checks to see if either card one is one position higher than card two, or vice-versa. If so, then it concludes that the player has a straight.

The second reveals the more pervasive kind of knowledge used by the program, which is namely applying knowledge to the results of a syntactic rule (in this case, the rule that checks whether the cards are pairs). Once we identify the value of the hand, we then try to rank it according to an index. Note that the index doesn't use certainty, although my initial inclination was to do so. However, I found that managing a largely arbitrary range of values was hard enough, when trying to merge them with other scales, let alone when a completely different scale is applied. Note that 60 is a largely meaningless number, and the knowledge comes mostly from the set of rules, which rank different possible hands relatively.
  ```

Design Reflection

Knowledge Acquisition

For my primary source of knowledge, I used a classmate who through part-time play of online Poker concurrent with MIT, has many thousands in less than two years. I took that to be a fairly indication of his expert status. What initially prompted my decision to focus on Poker was that he could play up to four tables at once, each a window on his desktop, and on each making decisions near instantly. Some decisions required considerably more thought, indicating that perhaps the betting decisions in Poker could be generally subdivided into two categories, very easy and trivial decisions (for the expert, at least), and more difficult and demanding decisions. While the first category has answers that can easily be arrived upon, it's not even clear that the second category has a correct answer that could be established by a consensus of the Poker expert community. This subdivision caused me to suspect that by nailing the easy questions, and not failing catastrophically on the hard questions, a simple system could perform well. I remarked to him that a program that could emulate his behavior must be possible, and from there the project emerged.

Conclusions to be Drawn

My intuitions about the problem are still accurate, I suspect, but what became clear was that the style of reasoning employed by my expert is incompatible with both the style of rule based reasoning used, as well as any easy systematic description. To formulaically integrate a number of differing metrics requires the adoption of a common scaling system, and doing so was completely foreign to the expert, complete guess-work for me, and arbitrary and meaningless for the both of us. Asserting that the playability of a hand depends on both the table situation and the hand strength is easy for the expert, and to give examples is easy for the expert. However, to actually quantify that relationship was

nearly impossible, and largely a source of frustration. Further, to make the situation more difficult, once the numbers were asserted, the actual evaluation of those numbers also became very difficult. When I walked through examples with a given number, there was no real metric for success or failure, other than the expert's notion of when a move was correct or not. However, the expert was inconsistent at best, often being unable to give a clear indication of why a recommendation was bad, less often being unable to even suggest a right move. "It could be played either way", was a quote I heard several times, and although such a quote spoke interestingly to the art of Poker, it wasn't useful for the scope of the project.

Although I anticipated this aspect of the problem, I didn't expect the degree to which it made systematic improvement to the ruleset impossible, especially with regards to table evaluation. The initial rules were a best guess numerical approximation of generalized advice given by the expert, and no good way to improve them existed. We could test against his judgment, but without the context of a real poker game he found the answers in general difficult to scrutinize. We could play it against real poker players or other poker machines, but between the huge number of playing styles and the degree of chance in the outcome, both approaches weren't particularly informative considering the small scale of limited testing I performed.

I am not particularly pleased with the results of the program, as I lacked the ability to truly leverage the ability of my expert. The quality of decisions made by the program is, I suspect, significantly worse than those made by novice Poker players. Additionally, the scale of the program is far limited from what I initially planned, which was to handle all rounds as well as a more robust betting system informed by certainties. To detail why my program fell short, I would point to three large problems.

1. My lack of success with the Joshua System. This was perhaps the most frustrating. Although LISP expressions are allowed, the manner in which I can take advantage of them is very limited. My final Betting assessment is a very good indicator of this. I wanted to assign intermediate variables based upon LISP

calculations, yet I could find no way to do this. This required ridiculous rewriting of my rules, and even the elimination of several variables, that were replaced by slightly modified copies of the rules inside other rules that referenced the eliminated variables. At one point I locked my Lisp Listener such that any button would result in an error message about presentation screens, preventing me from working. Although I still have no idea how that happened, I eventually figured out the correct dot file to delete to restore proper operation. The templates given to us by PS2 formed a nice starting ground, by routinely code that worked would break for seemingly unrelated reasons. After a certain point, I began rewriting rules rather than troubleshooting them. This created a very messy structure, further exasperated my attempts to determine meaning to a largely arbitrary valuing system, and made me spend about half an hour looking for a replacement implemented in Python or Ruby. No such luck.

2. Largely discussed elsewhere, difficulties with the representation and the acquisition of knowledge routinely raised their heads. Although I tried to solve them, I lacked a systematic approach to try to restore meaning to rules, and my expert was unable and unwilling to deal with the rule based implementation I had chosen, which I certainly can't blame him for as I sold the project to him as "hey, let's talk about Poker". My program has good results for a very limited range of possible inputs, and even those are a result of those very easy decisions that I discussed earlier.
3. Time was, of course, a limiting factor, although its not clear that any small number of hours would have resulted in a fundamentally better project.

I'm not sure how any of these problems would be surmounted except through better tools/better understanding of those tools, a far better method of evaluating program success (which is alone an immense undertaking), and possibly an expert more willing to engage in the technical nature of the project (I suspect that this was a relatively minor shortcoming, as I was willing to engage in the technical nature fully and still got nowhere particularly great).

Acknowledgements

David Smith Course 16 ‘04, Poker Expert.