Monte Carlo Methods for Decision Support

(or Winning More at Dealer's Choice Poker)

by Bradley Shanrock-Solberg, November 2015

Executive Summary

Background

Systems that contain intrinsic randomness are common, and are characterized by a long learning curve of "hard experience" before an individual can master such a system, gaining an instinct for the current environment and possible outcomes by actually using the system hundreds or thousands of times.

Lacking such a deep investment in learning a system the hard way, Monte Carlo Methods can provide insights and instincts into how a system behaves by exercising a model of the system in a way similar to a user and capturing key indicators that drive the decision making of the experienced individual who is successful at working with the system.

This case study will use the example of "Dealer's Choice" poker - a system where the dealer is allowed to vary the game played and number of wildcards to radically change the environment in which the game is played. Players who understand how the game is changed by the choices made by the dealer will perform better than those who rely on instinct or traditional casino/online poker models for how to bid and bet.

Issues

This technique requires a model that all of **intrinsic randomness,** where **inputs are easily automated** which, when aggregated can describe a **key decision process outcome**.

Results

While I have most often used Monte Carlo techniques to evaluate game design (eg the wildpoker R package and application), the techniques are equally applicable to engineering domains such as failure analysis of impact of counterfeit parts or political domains such as Nate Silver's predictions of USA Presidential and Senate outcomes. The approach is fairly common in any domain where the behavior of independent variables are modeled reliably but how they interact in a larger system is difficult to predict or calculate.

The Problem

Some systems are truly random. This is often obvious, as in the case of games which have random elements such as cards, dice, or computer generated damage and critical hit ranges. Sometimes the model is random, because the important result is dependent on lots of smaller results, each of which are expressed as a percentage. For example:

- Counterfeit parts in vehicles or structures are discovered to be present by inspection and are significantly weaker than the specced part. Each part has a different % chance to be counterfeit. Will the vehicle or structure have a catastrophic failure?
- A model has a good track record for predicting statewide elections based on state fundamentals and the history of polling for an election cycle. Will the United States Senate or Presidency change hands? Both depend on state-by-state results.
- An array of storage devices has a predictable failure rate for each device based on the age of the devices and the specific brand/model purchased. How likely is it that enough storage devices will fail in a short enough time period to risk the integrity of the array?
- A machine learning algorithm does not settle on a final answer, but cycles between several based on where it begins with your data. Is the algorithm useful anyway even though its output is not the same every time it is used?

Intrinsic Randomness at a low level of a complex system does not mean the larger system can not be understood. In absence of outside effects on the overall system system rules of thumb often emerge which help a novice (eg “Don’t Draw to an Inside Straight”. “Fundraising Numbers are Important in Elections”. “Broken Bones take 6 Weeks to Heal”. “Don’t get into a ground war in Asia”). True experts know when the rules of thumb don’t apply, generally based on an intuition gained from working with many cycles of the system over years. Those experts also have a plan for when the random numbers clump and the system enters a danger zone.

The goal of this exercise is to gain that intuition in minutes, rather than years, using the capability of modern computers to iterate over a model enough times to gain insight about how large numbers of independent, random events cause outcomes.

The Dealer’s Choice Poker Problem.

When playing poker, the player choices are limited to:

- Give up on the hand (fold)
- Stay in the game by paying money (call)
- Make everybody else pay more money to stay in (raise)
- In some games, choose which and how many cards to discard (draw)

Working strictly with odds, the way a rules-based computer might play the game, the basic concepts behind these decisions are:

- What hand I do I already have?

- How many cards are left in the deck that can improve my hand?
- Is the amount of money in the pot multiplied by the odds of improving my hand larger than the cost to stay in the game?

Skilled players also do their best to not play with a predictable pattern, and in addition are thinking about these concepts:

- Is my hand strong enough to win without improvement?
    - I want to keep everybody in the game
- What are the chances an opponent can win on a lucky draw?
    - I want to drive that person out of the game
- If my hand improves, how likely is it to win?
    - This modifies the calculation about betting cost vs size of pot
- What do I know about all the other players hands and their odds of beating me?

Coming in second best is really bad in Poker, it's where you lose the most money. On the contrary, luring people in when you know you have the best hand is how you win big.

A common thread is **"What is a good hand in this game?"** Without that key piece of information, the rest of standard poker skills are useless. The odds of improving your hand, for example, are irrelevant if the improved hand is still likely to lose. This dynamic is seen even in strict casino poker tournaments, requiring skill with 2-8 other players and often several games.

Dealer's Choice Poker is designed to make "what is a good hand" a hard question to answer.

- Changing the game changes the hand. A good hand in 5 card Stud is often a loser in 7 Card Stud. In Texas Hold-em if the community cards make most of a straight or flush, it is likely many players will have that hand, so having the best straight or flush is critical.
- Adding wildcards to the deck changes the value of hands significantly. Some games have wildcards that might emerge in play and a "good hand" definition will shift significantly once it is clear whether or not the extra wildcards will be in play.
- The number of players who stay until the last card also matters.. A two person game will require a weaker hand to win than a 6 player game. A 6 player game reduced to two players by forcing the other four to fold will still have stronger hands win than a two-person game, because the two who stayed in developed a decent hand or makings of one early, where weaker hands folded - if your hand is strong for 2 player but only middling for 6 player it'll help if some of your opponents never get to see all their cards.

People good at Dealer's Choice poker have a good feel for how poker variants change the strength of a hand. When they deal, they can often gain a significant advantage for that game simply by choosing a game that is very different in hand mix than the recently played games.

This expertise is normally only gained by a lot of play, to the point where even an unfamiliar game can be estimated - but the question of "what is a good hand" can also be answered via

Monte Carlo simulation - if the question is posed properly and the results are presented carefully.

Choosing a Question That Can Be Answered

While it is impossible in a system with intrinsic randomness to reliably predict a single outcome, if a decent model of the system exists, it usually can answer a question helpful in understanding what can affect the probability space.

In the case of a failure analysis problem, if you have a model of how the entire vehicle or structure behaves under stress, the impact of counterfeit parts can be built into the model and that model could be run enough times to simulate all the buildings or vehicles you are likely to ever build to see if random chance is likely to concentrate enough counterfeit parts on a single vehicle or structure to cause it to fail.

In the case of elections, a good model of each of 50 states can be run with the sampling of outcomes in all 50 states to measure electoral college counts or Senate seats.

In either case, the law of large numbers helps give a sense of the distribution, and the critical tipping point (eg failure of airplane or electoral control) can be used to get a probability - all of the measurements below the threshold measure probability of one outcome, all above the opposite.

The model's parameters can then be tweaked to see how they affect the overall odds. In an electoral model, for example, something like the President's popularity in polls might be a factor, and negative advertising could depress that, or actions by the President could improve the number, and the impact of a change in either direction on the overall election model might give insight on how to act to improve the odds of the desired electoral outcome.

The Dealer's Choice Poker Problem.

The true outcome of a game of poker depends on how the players behave, not strictly what cards are dealt. This requires choosing a metric which is conservative, but which provides guidance into correct behaviors.

Much of poker is mechanical - the deck is shuffled, it does or does not have wildcards and the hands are dealt in a way prescribed by the rules of the game. Each player who stays in the game for the showdown has a certain number of cards and tries to make the best hand. What is the best hand is also determined by the rules of a specific game - in some games a "high" hand is desired, in others "low" in others the pot is split based on criteria having little to do with normal poker hand values.

The "best" hand in a given deal of poker therefore is defined as "the hand that won if all players stayed in the game until the end". While it will not always match what happens in play,

especially for a hand that develops late in the draw, it is the hand to match or beat in order to be safe. This has a number of advantages for the model - it doesn't depend on player skill, and secondary considerations such as splitting the pot and multiple hands of same type can be managed.

Inputs Must Be Easily Automated

Monte Carlo techniques can be thought of as "heck with analysis, just measure it". In this respect they have some similarity to my approach for black box testing complex systems.

Ideally, you have a model that can predict a single event reliably, whether it is a fixed value (such as % of parts that are counterfeit, or odds that a Democratic Senator candidate can win in Texas) or requires sampling (such as rolling dice, drawing cards or sampling from a theoretical normal distribution). In this situation you can simply put a loop around your model, capture output in a data frame, summarize and graph it.

Sometimes the model is static but input data has a random element. That also is fine, as long as you can reliably generate the random data, and the generation algorithm matches reality well.

Dealer's Choice Poker is clearly a good candidate for this criteria - every element of the deal is easily simulated with a computer program, as long as player tactics are left out. Different games will deal differently and evaluate differently, which means more games will increase model complexity. Player tactics are a significant worry, as the modeling is much more difficult and far more likely to introduce bias, based on how the programmer prefers to play poker.

Analysis of Alternatives

At this point the thought processes in building the model will be explored using the Dealer's Choice Poker Problem.

There are four key variables that affect "best hand" outcomes in the poker model:

- How many players exist. There must be two players to have a contest, and no more than can be dealt a full had given the size of a deck (usually 52 cards). This is easy.
- Number of extra wildcards. This is relatively easily introduced at the time of the shuffle.
- Supported games. This is where most of the work in the model will reside, as while the basic rules of poker and hand value are fairly easy to code, many of the games have victory conditions or extra wild cards or additions/subtractions to hand size that require the model to understand the order the cards were dealt, which cards are "hole" cards and which cards, in the end, belong to each player.
- Player tactics. In Draw Poker and variants where the player has the option to pass cards, steal cards, bid on cards or otherwise interact with the deal player behavior would have to be modeled. As players vary widely in skill and have a strong interest in not

behaving predictably, any purely probabilistic set of player tactics will likely bias the model, although even a biased model is likely better than approximating the outcome based on less tactical games, but that approximation is still better than nothing.

I created a rapid prototype of the model incorporating the player and wildcard logic, but only supporting three games (5 card stud, 7 card stud, Texas Hold-Em). This took only about two days to put together and test. I gained the following insights:

- R is too slow to perform the hand value logic in real-time for hundreds or thousands of simulations, although it is fast enough to pre-calculate the numbers. I had visualized something with sliders to let a user rapidly evaluate changes based on player, wildcard and game, and the result
- The problem of competing hands is significant - some games are likely to end up with everybody having, say, a Flush, and in such situations it is extremely important to know if you have the best hand of that type. I attempted to show this with “high flush” vs “flush” bars but I was not happy with that approach and needed to improve the visualization.
- Most of the model complexity was in the supported game logic, and to expand to the list of games I wished to explore, I had to exclude player tactics, which would both be expensive in time to code and less reliable than simply supporting every type of game that did not rely on player tactics.

My rapid prototype logic could already support an infinite variety of wildcard combinations and about half of the commonly used Dealer Choice combinations such as “Deuces”. It was a simple matter to resolve to include the others I’d encountered or heard of. Likewise the player variable was pretty thoroughly nailed down. What remained was the type and variety of games I could support - at a minimum any game used in professional poker competitions and any non-tactics based game I’d encountered in my years of playing “Dealer’s Choice” poker were to be included. Beyond that any variation that didn’t require new logic was included, and a few more got with only minor changes required also got included and in one case (The Good The Bad and The Ugly) I added the game simply because it had a discard rule that did not rely in player tactics, so I could build the framework for future draw/discard/pass games should I someday upgrade the model to include player tactics.

In the end, over 50 games were supported, and I had to include all of the following into the overall model logic:

- Split hand logic (as in Hi-Lo, Chicago, Count your Diamonds, Spots)
- Low Hand Wins logic (Lowball, Razz) with several variations (A-5, A-6, 2-7)
- Omaha-like logic where you can make a hand out of only part of hole+community cards
- Multiple community layouts, as in Iron Cross or Double Flop Hold-Em
- Floating wildcard logic such as Follow the Queen or Dirty Schultz
- Hand-specific wildcard logic such as Kankakee or Little Ones
- High hands with less than 5 cards as in 3 Card Monte or Hurricane

In contrast to the 2 days to make the prototype, all of these variations and exceptions took nearly 3 weeks to code and thoroughly test, with the “low hand” logic combined with “aces can be low or high” causing the most quality assurance headaches.

Solution Approach

In addition to simply modeling the games, other functions had to be written to perform the Monte Carlo method of playing many games and capturing statistics on their results in a form suitable for visualization. Some of the game variants result in variable numbers of wildcards, so the progress of wildcards also had to be tracked. Because some games split the pot, and ties are increasingly possible as wildcards increase, the percent of the pot won by a winning hand also had to be tracked. (decisions on betting are made by comparing the cost to draw to the size of the pot you might win if your draw prospers, so if the pot will be fragmented this needs to be part of the decision process)

The R package exposes only three functions, and the application primarily uses only wpgraphs.

- **wpgame -** plays a single game, shows the card layout and which hands were used by each player to try to win, plus some basic summary statistics. Used to show that the model is following the rules and for debugging, beyond its use as a building block of the later functions
- **wpstats -** plays as many games as desired, with option to reset the default random number seed. Organizes data either into a raw data frame or a summary “gstat” object used by the reporting module.
- **wpgraphs -** shows a set of graphs tied to a “gstat” object, which vary a bit depending on whether or not the game has pot splitting or variable wildcards. While the function produces a canned 4x4 array of graphs, it will alternately return the raw graphics object to allow generation of other graphs based on the same data. This is important because the package also includes precalculated gstat data for every supported game, (see Implementation, below) and accessing the raw gstat list for a given game instead of publishing the 4x4 graph can be done with a simple parameter change to “TRUE”.

Implementation

Changes from Initial Approach

The rapid prototype imagined a user would only interact with the application. Converting the model to a R package meant that there had to be tools allowing others to use the model from the R console, and to validate the results.

The hand evaluation program took the bulk of the time, about .006 seconds per hand to run. This means 1000 games of 7-player, Seven Card HI-LO means evaluating 14k hands and takes about 90 seconds to run, and a game like Courchevel Hi-Lo for the same number of players

means evaluating 980000 hand combinations, or a bit under 2 hours to run. This is clearly unacceptable for a decision support application, so a utility function was written to loop through all supported game, player and wildcard combinations, limited only as follows:

- Player maximum was 8 even if the game allowed more - it's hard to physically fit more around a table, and this matches casino tournament limits.
- Wild card combinations were from zero to 7, and were made up only of combinations of Deuces, Suicide King and One-Eyed Jacks. Beyond 7 additional wildcards results tend to skew very strongly to maximum strength hands unless the game is very limited, which is why you see Pregnant Threes and Dr Pepper (both 12 wildcard combinations) normally only called in 5 card draw poker.

The approach of capturing Monte Carlo simulations for all combinations allowed about 5 days of processing to be compressed into a list with a few thousand rows, each row corresponding to a legal game, and each a valid target for the **wpgraph** program.

The Shiny application was modified to turn player and wildcard input into a slider, and the supported games into a radio button allowing both the game and equivalent variants to be visible at a glance. The change to simply pointing wpgraph at a list instead of actually playing 100-1000 games made the application performance instant.

Lessons Learned

The game variations were captured in a single data frame, and were treated as inputs to switching logic among the basic functions which shuffled the deck, dealt the cards and evaluated player cards to find the best main hand and best split hand (where required).

This architectural decision allowed a great deal of flexibility in adding new game variants. If a new row could be added for a new game, it is likely little or no new coding would be needed. If new logic was needed, it would be captured in a new column and affect either the deck manipulation or hand evaluation logic, sometimes both. A few variants violated the assumptions of the basic framework and would require significant rework to include, even though they did not involve player tactics. Those variants were left out of the 1.0 release.

Graph structures also had to be redefined for each split hand variant that deviated from a standalone game. Low-hand variants in HI-LO games can fail to have anybody win the low half if hands aren't good enough (so the Main Hand winner gets it all), and some of the other split variants also have the possible outcome of no legal split hands. Any hand structure where legal values didn't match normal high or normal low order or legal hands required extra graphic work.

Results

Questions Answered

With the new application, it is very easy to see the strength of a player's hand for a particular game. It shows whether that player might have rivals for the best hand, whether or not the "best hand" matches the wildcard situation, whether ties or a split pot might reduce the winnings..

Insights Gained

With a decision support tool like the Wild Poker application (or wildpoker package, which can do the same thing from the R command line), it is possible to quickly familiarize yourself with whichever games you expect to encounter and either memorize or make a quick cheat-sheet of how hands shift as players arrive or leave the table, or as the game called changes.

This will reduce the chance of paying into a losing hand, one that will lose even if you draw out an improvement to the hand. It will also let you know whether or not it is to your advantage to drive rival players out of the game, or try to keep them in the game only to be defeated by you - because you know you've already made a hand that is quite strong in this environment.

Similar tools, such as for the failure analysis of vehicles or buildings with counterfeit parts, or electoral analysis of national elections based on statewide models give a sense for "Where you are" in even when at the bottom level, your overall system has random elements.

Actions Taken

Sometimes you will be in a position to affect destiny, and want to know the likely impact of your actions to the overall range of outcomes. A change in aircraft design perhaps, or a new round of political advertising. Or in the case of Dealer's Choice, you are the dealer.

While I lacked the Wild Poker application, I've been using similar if sometimes less sophisticated Monte Carlo techniques to gain insight into games for most of my life, including for Dealer's Choice poker. My approach to Dealer's Choice when it was my deal was to call a radically different game, where a "good hand" was nothing at all like what it had been in recent games.

Players who did not have an intuition for the shift in "good hand" would make mistakes well below their usual level of play, and that, combined with avoiding similar errors when other people called the significantly different games meant that in 25 years I've never walked away from a "friendly" Dealer's Choice game behind, and was often one of the table's big winners.

Conclusion

Monte Carlo techniques are the first tool I reach for when evaluating game design, whether because I wish to participate in the game and play at a high level even as a newcomer, or as an abstract exercise, comparing different games evaluating a new game's possibilities.

I have found it useful in other arenas which contain either random elements in the model or in the input data, because truly random behavior is not intuitive. Random events "clump" and causes "outlier" events that are often the most interesting (or risky) thing about the system.

When you have a good model of many small events that together predict behavior of a larger system, it is often very hard to calculate the impact of those events but often quite easy to simply measure them with Monte Carlo Techniques. This approach provides intuition about the behavior of the larger system normally gained only with long experience and allows some ability to predict the impact of changes to either the macro environment or individual components on overall behavior of the system.