CSE P 573 Program Assignment 3

Alan Ludwig alanlu@uw.edu (worked in partnership with Shawn Callegari)

State Space

Programming assignment 3 was to create a program that determined the optimal policy for a simplified version of blackjack parameterized by the probability of getting a 10. To solve the problem I first generated all of the interesting states for the problem. The definition of the state space I used was nearly the one provided in the problem statement. I simplified the state given by changing the integer variables of NumAces and dNumAces to Boolean variables that indicated if the hand had ANY aces. It isn't necessary to know how many aces there are in the hand. Only one ace can be counted as 11 without busting. With that in mind, I generated a bit less than 17,000 states.

Modeling Costs and Rewards

Once the states were generated the next job was to assign values. There are two sets of values to keep track of. The first is the bet, or the cost to play. The second is the possible payouts. I chose to ignore the cost of the initial bet in my model. And so I valued the terminal states based on the payout without respect to the cost. So the final value for blackjack was 2.5, rather than the 1.5 profit. The exception to this was in calculating the expected value of double and split. When calculating the value of double and split the calculated value was 2 * value - 1 to account for the additional $1 bet that must be made when splitting or doubling down. The 2x is to indicate that we're playing two hands now so whatever the expected result is, it is doubled. This approach is consistent with the idea of "expected value". We're going to place our bet and in the end we will attempt to maximize the value we receive back for that bet.

Doubling Down and Splitting

Once each of the terminal states had been valued based on the rules, the expected values of the rest of the states for the "stand" and "hit" states were calculated. Double down and split only applies to initial states so the optimal strategy could be calculated on intermediate states with hit and stand only. Hit and stand values are calculated first for the initial states. Double down was calculated by calculating the expected value by taking a single card and then standing. Spit was more interesting.

Split was calculated using iterative evaluation. An arbitrary value was chosen as the initial value for the split action. Then, the expected value for the state was calculated. That calculation is self-referential, and includes its own expected value. On each iteration, the estimate is improved and the process is iterated until the amount of the correction on each iteration is less than epsilon (which I chose to be 1.0e-6). The result is an expected value for the splitting action that is arbitrarily close to the actual expected value.

Unbounded expected value for 1010 at p > 0.5

In testing I experimentally tried p = 0.5. I found that instead of taking a few seconds to complete, the program appeared to hang. For the value p = .49 the program finished executing in a few seconds. p=.499 took a little longer and p=.4999 even longer. The evaluation of Hit, Stand, and Double Down complete in essentially linear time for a single case (assuming all dependent cases were previously calculated). Only the evaluation of the split cases could be the culprit. What I found was that the expected value of the 1010 (a pair of 10s) had a critical value at p=0.5. I experimentally found that the expected value of splitting is unbounded at that point. I found cases where it was unbounded in the negative and positive direction. Instead of converging, the expected value diverged. Eventually the program would complete as the expected value eventually became *infinity* or *–infinity* and then the delta calculation would subtract the values and become *NaN* . Any comparison with NaN is false and therefore through the magic of floating point numbers my while condition would be unsatisfied and the calculation would terminate. This happened more quickly for values above p=0.5. This would give the false impression that everything worked normally for p=.75 for example. But it appears that for all cases where p > .5 the case of splitting for 1010 is unbounded.

Allowing Casino Play for p>.5

It goes without saying that having an unbounded positive expected value would be unfortunate for the Casino. Of course, infinite payout would only be possible if you had infinite money to bet. If you wanted to play such a game in a casino, applying special rules to 1010 like are applied to AA would be the start of rehabilitating such a game. Of course, some additional changes would need to be made to ensure that the house made an acceptable margin over the long term. Casinos are perfectly capable of paying out large sums of money, as long as the expected value of the game is always in favor of the house. An interesting extension to the assignment would be to also calculate the expected value of playing the optimal strategy. It would be interesting to calculate the largest value of p for which the house still makes some money. Or stated differently, for what value of p is the expected value of a $1 bet played optimally give you all your money back.

Calculating Optimal Policies In The Case Of Unbounded Rewards

I'm sure if I spent a little time, I could apply the theories of infinite sequences and explain exactly why p=.5 is a critical value for splitting 1010. But on reflection in this context I think a better take away would be to appreciate the importance of gamma in the theory of MDP. By choosing an exponentially decaying gamma you can (by those same theories of infinite sequences) discount even infinite future rewards enough and determine a rational optimal strategy that maximized not only the future rewards but the rate at which they were acquired. I believe such a strategy could likely also be applied here so that an optimal strategy could be determined even in the event of infinite future reward (or cost as the case may be).