

BPP REPORT

(CSE1307 ADVANCED PROJECT 1)

By Robiah Elsaadi 20486162

- Introduction
- Texas Hold Em Rules
- Poker Bluffing Strategies
- Poker Sandbagging Strategies
- Pot Odds Strategy
- Bayesian Networks
- Coding Implementation to BPP
- Results
- Conclusion
- References

Introduction

The purpose of this project was to be introduced to an artificially intelligent poker player using a Bayesian network for decision making (BPP – Bayesian Poker Player). We had then been given the task of researching different sandbagging and bluffing strategies, and then implemented them into BPP's existing code. We had then conducted tests on the changes we made, and compared the results with that of the previous BPP's code , observing any effects and improvements made (if any).

Texas Hold Em Poker Rules

Texas Hold Em Poker is a variation of poker where each player is dealt 2 cards face down, and they then play to make the best hand out of the community cards – consisting of 5 board cards revealed over 4 rounds, the Pre-Flop, the Flop, the Turn, and the River. The Pre-Flop, is the round where each player has bet, and decides to play with the hand they have been dealt. The Flop round then reveals the first 3 community cards on the board, where each player then decides if they wish to bet and continue playing. The Turn round then reveals the 4th community card, and again players decide whether they want to bet and continue playing, or fold. The River round then reveals the final 5th community card, and again the remaining players decide whether to bet or fold.

The objective for each player is to try and make the best hand out of the 2 cards they have been dealt with the community cards. The player with the best hand at the end of the game wins the

accumulated pot.

Poker Bluffing Strategies

The bluff – a strategy where you bet strongly with a weak hand, deceiving opponents and leading them to believe you have a strong hand.

BPP currently bluffs by generating a random number, and comparing it with a bluffVector (which is an array of values that represent a threshold for each round). If the random number is less than the value in the round position in the bluffVector array, then it changes to bluffing mode. While bluffing, BPP looks at the current board cards, and then gets the most probable hand higher than its current hand.

Some popular bluffing strategies are:

Position of the dealer button: It is wise to take advantage of your position to the dealer button. You are at a great advantage when you are closer than others to the button, and at a greater advantage when you are the dealer, as the button makes its way around the table each game. Because the dealer acts last, they have the advantage of observing each other players action, and they can then take the opportunity to bluff. In a situation where all players have checked, leaving the dealer last to act, the dealer has the chance to then bluff and bet, in the aim of leading others to fold – and winning the pot.

Bluff failed, Continue or not? You may win some games successfully from bluffing, but there are games when another player continues to call your bluff – they are confident their hand is the strongest. This is going to lead to a costly mistake if you have been betting aggressively, and fail to examine what the other player may have. You may be faced in a situation where you will need to decide if you should continue to bet, or fold and cut your losses. At these times it is important to examine the community cards carefully. If they call and you suspect they have a strong hand, it is best to give up. Otherwise if you believe you have a good chance, continue bluffing.

This strategy should also be applied for when you are up against a player who seems to call every round – a term for this type of player is a “calling station”.

Poker Sandbagging Strategies

Sandbagging – Where you bet weakly with a very strong hand, in an attempt to lure opponents until the final round. The aim is to maximise the pot winnings for you to collect when you have won the game.

BPP currently does not sandbag, so my task was to implement this type of play.

Some popular sandbagging strategies are:

Strategy #1: Check and call during pre and post flop, and then gradually raise the bet during the turn. Then at the final round proceed to increase the bet drastically – in an attempt to either lead

others to fold, or cause an all-in situation where you eliminate others from the table. If it is 'all-in,' be sure that you have the strongest hand possible.

Strategy #2: Rely on a raise from your opponents. You can also combine this behaviour with strategy #1. Sandbagging is most effective when you suspect opponents will either:

1. Raise when you check or call.
2. Fold when you raise.

After an opponent raises, if there are remaining players to act after you, then call that opponents raise, with the hope that the remaining players will either call or raise. You don't want to bet strongly, as if you did and were to raise in this situation, then there is more of a chance that the other players will fold – which is not what you want early in the game, as you want to maximise the winnings in the pot.

Pot Odds Strategy

The pot odds tell you if it is worth calling or not. It is the ratio of the cost to call and the current pot size. You then compare the pot odds with the probability of winning, based on your hand and the current board cards. If the pot odds are greater than your chances of winning, then it is not worth betting – and the converse if the pot odds are lesser than your chances of winning, then it is worth betting.

This can be applied to bluffing, where you over represent the hand you have, while also calculating the cost to call each round.

Sandbagging on the other hand, you would want to call each hand, but increase the amount of your bets slowly, towards the end of the current game.

Bayesian Networks

A Bayesian network is a probabilistic model used in decision making, and artificial intelligence. The network consists of nodes acting as variables, which may have dependencies on the values of other nodes. As information is fed into a node below, the other related nodes above will then update their states based on this given information and state of the updated node.

A node can represent either:

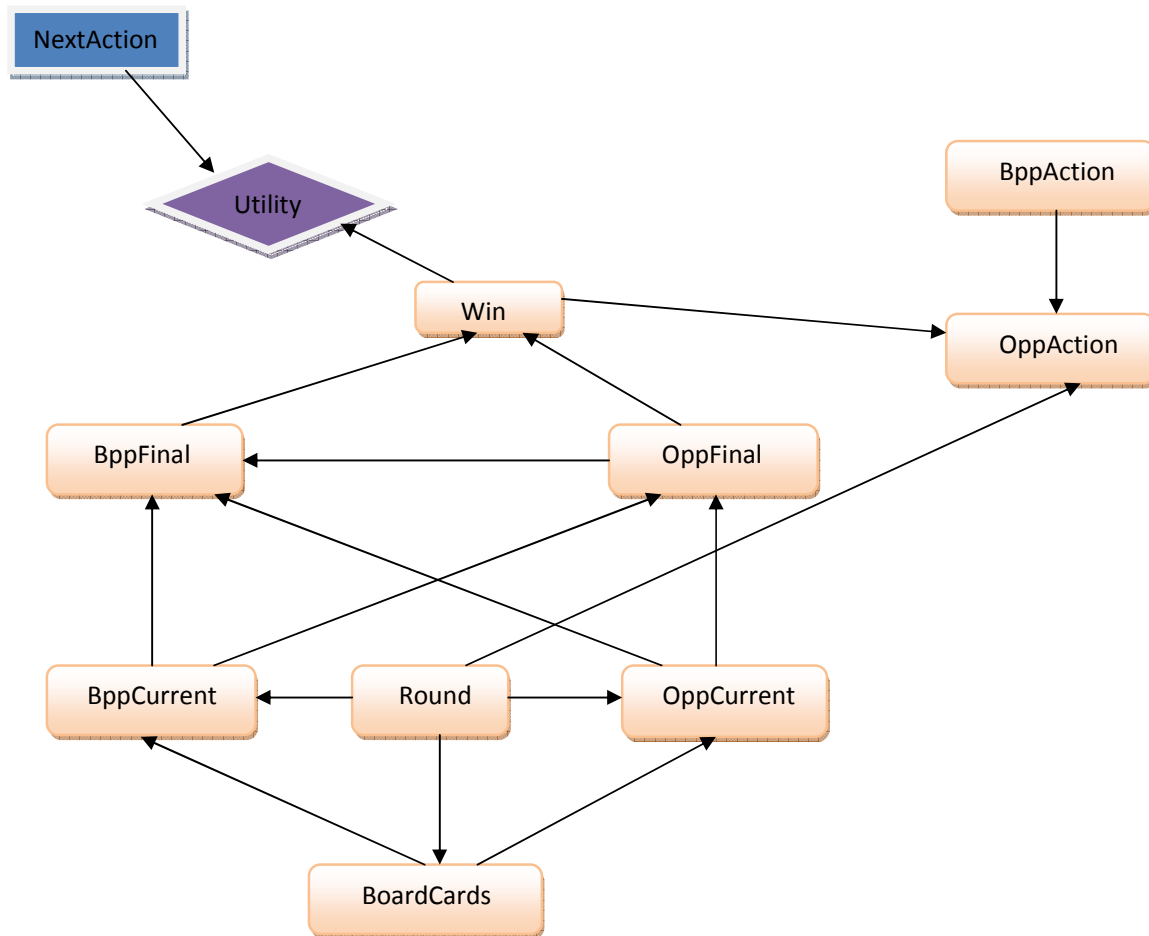
- Boolean Values: T or F for a certain state.
- Ordered Values: A level of magnitude indicating *high*, *medium*, or *low*.
- Integral Values: Consisting of a range of possible values.

BPP uses a Bayesian network to determine its probability of winning each round. It inspects the board cards, and its dealt hand, and proceeds to retrieve information from the network, such as what best hand it can make out of the board cards, and also what the opponent is likely to have, and what their action in the round might be.

Through method and API function calls, we can retrieve information on BPP's beliefs of winning,

what it believes its final hand will be, and what it believes the opponent's final hand will be.

Here is the current diagram of the Bayesian network used for BPP:



This network is used by BPP to make decisions on the action it will perform each round. Using API calls, we can retrieve information on the current state of any of these nodes. One node in particular that I use in my sandbagging and bluffing strategies is the Win node. The state of this node will show BPP's probability of winning, losing, or drawing. The diagram above shows that this node retrieves information from the nodes below it, to change its state between whether it thinks it will win, lose, or draw.

Coding Implementation to BPP

BLUFFING

I made the following changes to the attributes in BPP class:

`#bluffVector = [0, 0.03, 0.07, 0.15] - old bluffVector`

```
bluffVector = [0, 0.18, 0.24, 0.3] #New bluffVector I created
bluffingThreshold = [0, 0.3, 0.4, 0.5] #A threshold for deciding whether to bluff, to be compared with
the winBeliefs vector
```

The coding implementation for bluffing in the calcAction() method is as follows:

```
...
#Updated bluffing 09/2007
if not self.bluffingOff:
    if self.bluffing:
        #An idea I had was to cause BPP to fold if the opponent seems to call BPP's bluff until
        #the final round. The following lines achieve this by keeping count of BPP's bluffs.
        if oppAct == 'l' or oppAct == 'r':
            self.bluffCount = self.bluffCount+1
        #is the opponent continuing to call BPP's bluff? Then
        #turn bluffing off and let BPP decide what action to
        #do
        if self.bluffCount > 3 and if random.random() > 0.75
        and round == THG.numRounds-1:
            self.bluffing = 0
            break
        #else bluff and turn sandbagging off
    else:
        myCalculatedState = self.doBluff(round)
        self.sandBagging = 0
    #if not already bluffing, BPP checks if it doesn't have a good chance of winning.
    elif winBeliefs[0] < self.bluffingThreshold[round]:
        #if it doesn't, then randomly decide to bluff, and if bluffing then turn off sandbagging
        if random.random() < self.bluffVector[round]:
            if APP.showthinking: dout(3, self.username+": I'm bluffing")
            self.bluffing = 1
            myCalculatedState = self.doBluff(round)
            self.sandBagging = 0
...

```

SANDBAGGING

A sandbagging algorithm:

1. If your hand is strong, begin sandbagging.
2. Check, or call any bets, during the first round
3. After the turn is revealed, if it is your action first then check. Else call the opponents bet.
4. After the river is revealed, if it is your action first then check. Raise your opponents bet either way.

Implemented in the calcAction() method:

```
...
#compare the P(Win) against the threshold (if greater than 65%)
#and if greater, and is not already bluffing, then turn on sandbagging
if winBeliefs[0] > 0.65:
    if not self.bluffing:
        self.sandBagging = 1
        self.bluffing = 0
        if APP.showthinking: dout(3, self.username+": I'm sandbagging.")
        #if its pre flop
        If round == THG.PREFLOP:
            #BPP to bet or call any other bets

```

```

        if action == 'r':
            return 'b'
    #if its post flop
    if round == THG.POSTFLOP:
        #if BPP wants to raise or bet, override action and check instead
        if action == 'r' or action == 'b':
            return 'c'

    #if its the turn
    if round == THG.TURN:
        #if BPP wants to raise, override action and check instead
        if action == 'r':
            return 'c'

        #if opponent has checked, then BPP will bet
        if oppAct == 'c':
            return 'b'

        #else if opponent has bet, then call
        elif oppAct == 'b':
            return 'l'
        elif oppAct == 'r':
            return 'l'

    #else if it's the river, raise aggressively
    if round == THG.RIVER:
        return 'r'

```

...

Results

Results show the performance of BPP playing 20 sessions, with 1000 hands each session.

The first test case was conducted with the previous version of BPP, which did not include the changes I had made. In this test, BPP played against itself.

Before changes:

Opponent: Bpp 3.1.4

```

bpp3_1_4 20:1000 Session: 1 - ($-2725.00 | -0.27 sbu | 0.00% | 464.00)
bpp3_1_4 20:1000 Session: 2 - ($-3845.00 | -0.33 sbu | 0.00% | 470.50)
bpp3_1_4 20:1000 Session: 3 - ($-1225.00 | -0.26 sbu | 0.00% | 475.33)
bpp3_1_4 20:1000 Session: 4 - ($-500.00 | -0.21 sbu | 0.00% | 479.25)
bpp3_1_4 20:1000 Session: 5 - ($185.00 | -0.16 sbu | 20.00% | 478.00)
bpp3_1_4 20:1000 Session: 6 - ($-3725.00 | -0.20 sbu | 16.67% | 476.83)
bpp3_1_4 20:1000 Session: 7 - ($-650.00 | -0.18 sbu | 14.29% | 479.71)
bpp3_1_4 20:1000 Session: 8 - ($-4480.00 | -0.21 sbu | 12.50% | 480.13)
bpp3_1_4 20:1000 Session: 9 - ($-1085.00 | -0.20 sbu | 11.11% | 479.89)
bpp3_1_4 20:1000 Session: 10 - ($1710.00 | -0.16 sbu | 20.00% | 480.50)
bpp3_1_4 20:1000 Session: 11 - ($-1365.00 | -0.16 sbu | 18.18% | 480.64)
bpp3_1_4 20:1000 Session: 12 - ($650.00 | -0.14 sbu | 25.00% | 481.00)
bpp3_1_4 20:1000 Session: 13 - ($525.00 | -0.13 sbu | 30.77% | 481.00)
bpp3_1_4 20:1000 Session: 14 - ($-7925.00 | -0.17 sbu | 28.57% | 479.07)
bpp3_1_4 20:1000 Session: 15 - ($-3050.00 | -0.18 sbu | 26.67% | 479.47)
bpp3_1_4 20:1000 Session: 16 - ($1570.00 | -0.16 sbu | 31.25% | 480.25)
bpp3_1_4 20:1000 Session: 17 - ($255.00 | -0.15 sbu | 35.29% | 481.00)
bpp3_1_4 20:1000 Session: 18 - ($-600.00 | -0.15 sbu | 33.33% | 481.11)
bpp3_1_4 20:1000 Session: 19 - ($975.00 | -0.13 sbu | 36.84% | 480.95)
bpp3_1_4 20:1000 Session: 20 - ($-30.00 | -0.13 sbu | 35.00% | 481.25)

```

20 Sessions, 1000 Hands
Sessions won: 35.00% (7)
Sessions won t(19)-value: -1.34
Earnings: -0.13 sbu (\$-1266.75)
Earnings Sd: 0.24 (\$2393.88)
Earnings t(19)-value: -2.31
Hands won per session: 48.13% (481.25)
Showdowns per session: 57.95% (579.50)

These results show that before the changes I made, BPP had won 48.13% of hands per session, and had one 7 sessions out of the 20 (35%).

The second test conducted was with my updated version of BPP, versus the previous version of BPP. This way, we could see if there was much of an improvement in BPP's actions and behavior.

After changes:

Opponent: Bpp 3.1.4

bpp3_1_4 20:1000 Session: 1 - (\$-2980.00 | -0.30 sbu | 0.00% | 492.00)
bpp3_1_4 20:1000 Session: 2 - (\$3350.00 | 0.02 sbu | 50.00% | 493.50)
bpp3_1_4 20:1000 Session: 3 - (\$-2610.00 | -0.07 sbu | 33.33% | 490.00)
bpp3_1_4 20:1000 Session: 4 - (\$-4235.00 | -0.16 sbu | 25.00% | 483.00)
bpp3_1_4 20:1000 Session: 5 - (\$-895.00 | -0.15 sbu | 20.00% | 482.80)
bpp3_1_4 20:1000 Session: 6 - (\$-1870.00 | -0.15 sbu | 16.67% | 484.17)
bpp3_1_4 20:1000 Session: 7 - (\$-5315.00 | -0.21 sbu | 14.29% | 479.00)
bpp3_1_4 20:1000 Session: 8 - (\$-3055.00 | -0.22 sbu | 12.50% | 478.75)
bpp3_1_4 20:1000 Session: 9 - (\$5655.00 | -0.13 sbu | 22.22% | 482.44)
bpp3_1_4 20:1000 Session: 10 - (\$1175.00 | -0.11 sbu | 30.00% | 483.40)
bpp3_1_4 20:1000 Session: 11 - (\$-2345.00 | -0.12 sbu | 27.27% | 482.64)
bpp3_1_4 20:1000 Session: 12 - (\$2485.00 | -0.09 sbu | 33.33% | 483.83)
bpp3_1_4 20:1000 Session: 13 - (\$-4000.00 | -0.11 sbu | 30.77% | 482.15)
bpp3_1_4 20:1000 Session: 14 - (\$-3435.00 | -0.13 sbu | 28.57% | 481.50)
bpp3_1_4 20:1000 Session: 15 - (\$-4345.00 | -0.15 sbu | 26.67% | 481.07)
bpp3_1_4 20:1000 Session: 16 - (\$-4660.00 | -0.17 sbu | 25.00% | 479.56)
bpp3_1_4 20:1000 Session: 17 - (\$-2710.00 | -0.18 sbu | 23.53% | 479.47)
bpp3_1_4 20:1000 Session: 18 - (\$-6485.00 | -0.20 sbu | 22.22% | 477.33)
bpp3_1_4 20:1000 Session: 19 - (\$1080.00 | -0.19 sbu | 26.32% | 477.42)
bpp3_1_4 20:1000 Session: 20 - (\$-5715.00 | -0.20 sbu | 25.00% | 476.00)

20 Sessions, 1000 Hands
Sessions won: 25.00% (5)
Sessions won t(19)-value: -2.24
Earnings: -0.20 sbu (\$-2045.50)
Earnings Sd: 0.32 (\$3242.02)
Earnings t(19)-value: -2.75
Hands won per session: 47.60% (476.00)
Showdowns per session: 57.78% (577.85)

Results show that there was no significant improvement in BPP's actions, as it had only won 5 out of the 20 sessions played (25%), and had won 47.6% of hands per session.

Conclusion:

The changes I had made to BPP did not make any improvements judging from the results (although the nature of the game is random) I did however gain an insight on AI, and decision making using Bayesian networks. An idea for a future improvement could include creating a “strategy” node in the Bayesian network, which would include values for whether it should play normally, bluff, or sandbag. This node would retrieve information from other nodes, for example its probability of winning, what the opponent’s last actions were, or what hand the opponent could expect to have. The node could also choose the most probable hand higher than its current one (if bluffing), or avoid raising in early rounds (if sandbagging). That way, the code could retrieve this information from the network directly, which would reduce code, and also improve its decision making even further.

References:

Used throughout my research of this project:

TightPoker Website - Bluffing:

<http://www.tightpoker.com/bluffing.html>

How to play poker guide Website – Sandbagging:

<http://www.howtoplaypokerguide.com/sandbagging.shtml>

Slow Play(Poker)/Sandbagging Wiki:

http://en.wikipedia.org/wiki/Slow_play_%28poker%29

Bluff (Poker) Wiki:

<http://en.wikipedia.org/wiki/Bluffing>

Texas holdem-poker website - An Explanation of Texas Holdem Odds:

<http://www.texasholdem-poker.com/odds>

Poker tips website - Pot Odds:

<http://www.pokertips.org/strategy/pot-odds.php>

Pot Odds Wiki:

http://en.wikipedia.org/wiki/Pot_odds

Bayesian Artificial Intelligence (2004) – Chapter 2:

Kevin B. Korb and Ann E. Nicholson, Chapman & Hall/CRC Press