A No-Limit Omaha Hi-Lo Poker Jam/Fold Endgame Equilibrium

A No-Limit Omaha Hi-Lo Poker Jam/Fold Endgame Equilibrium

Kenneth Ho

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

November 2015

Abstract

Omaha Hi-Lo Poker is a variant of the game of Poker, with more possibilities resulting from having four cards per player and a split Hi and Lo pot, compared to Texas Hold'em Poker. Recently published algorithms, commodity cloud computing, and graphics processor acceleration enable the analysis of more complex games. We use the newly published CFR+ algorithm, OpenCL heterogeneous computing framework, and Amazon Web Services cloud computing to analyse Omaha Hi-Lo Poker. Using these tools, an endgame jam/fold ϵ-Nash Equilibrium is found and a scoring heuristic that approximates this equilibrium strategy is constructed.

Acknowledgements

I would like to express my gratitude to my thesis supervisor, Eric Giesecke, for his expertise in distributed computing and generous guidance.

It is my privilege to thank my wife Melanie for her patience and understanding through the late nights, blank stares, and quiet dinner tables. I couldn't have done it without you.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

Poker is a very popular and challenging card game for both humans and computers. John von Neumann, the creator of game theory, found poker to be an excellent starting point for analysing the behaviour of rational and unpredictable people. "Real life is not like [chess]. Real life consists of bluffing, of little tactics of deception, of asking yourself what is the other man going to think I mean to do. And that is what games are about in my theory." von Neumann went on to make many contributions to mathematics, computer science, economics, and physics, but he lacked the computing power necessary to solve a game as deep as poker. With the explosive popularity of online poker, inexpensive computers, and recent developments in game theory, researchers can now build strategies that approach perfect play.

The majority of poker research focuses on a variant of poker, Texas Hold'em. This variant is the most popular, as the game commonly seen on television and the format of the World Series of Poker main event. As the most heavily researched variant, computer scientists and researchers have built extremely strong strategies. In early 2015, the University of Alberta has even published an "essentially perfect" strategy for two player Limit Texas Hold'em, a game that is widely played in both casinos and online poker sites. And while there are no published perfect strategies for No-Limit Texas Hold'em, researchers have made large strategic contributions to this variant. With a few additional restrictions on the game rules, two and three player No-Limit Jam/Fold Texas Hold'em has been solved.

In comparison, Omaha Poker and its variants are somewhat less popular and lacking in research. This thesis project seeks to change that, by rigorously analysing the Omaha Hi-Lo Poker variant and computing an unexploitable strategy for No-Limit Omaha Hi-Lo Poker. The project uses game restrictions of jam-or-fold action constraints and two players to make the computation feasible, similar to the restrictions in No-Limit Texas Hold'em research.

1.1. Contributions

This thesis involves aspects of game theory, machine learning, and high performance computing. The major contributions of this thesis are:

(a) the computation of an ϵ-Nash Equilibrium strategy for Omaha Hi-Lo Hold'em Poker in the restricted domain of two player jam/fold games, and

(b) the creation of the ORACLE Strategy, a new evaluation heuristic for a hand in Omaha Hi-Lo Hold'em Poker, approximating a Nash Equilibrium, that a player can use to decide whether to jam or fold their hand.

1.2. Thesis structure

Chapter Two will further describe concepts in Omaha Hi-Lo Poker, game theory, and the computation frameworks used by this project. It will also highlight recent related work in game theory and poker research.

The following chapter describes the programs and algorithms used in this project. It steps through the questions that a poker player must answer to formulate a strong strategy, including the need to bluff, deceive, and consider the opponent's options, as demanded by von Neumann.

Chapter Four provides an analysis of the output from the project's poker programs, illustrating the steps taken to construct the ORACLE Strategy heuristic. Chapter Five concludes the thesis with a discussion of the project's limitations, and suggestions for further research.

Chapter 2: Background

This chapter begins with an overview of Omaha Hi-Lo Poker, the rules of the game, and definitions of poker game concepts, followed by a description of a poker tournament and strategic aspects of the tournament structure. Next, this chapter covers basic concepts in game theory. It continues with a discussion of previous work in computer poker and tournament poker strategy, with a focus on the Counterfactual Regret Minimization (CFR) algorithm and its relationship to Nash Equilibrium strategies. The chapter concludes with a discussion about the Amazon Web Services platform and OpenCL heterogeneous computing framework that I used to search for an equilibrium strategy for a two-player Omaha Hi-Lo Poker tournament endgame.

2.1. A No-Limit Omaha Hi-Lo Poker Jam/Fold Tournament

2.1.1 No-Limit Omaha Hi-Lo Poker

Omaha Hi-Lo Poker is a variant of Poker that uses a standard fifty-two card deck, consisting of cards with a suit and a face value. Cards can have one of four suits: Clubs ♣, Diamonds ♢, Hearts ♡, or Spades ♠. Each suit appears on thirteen cards, with face values of thirteen different values, in descending rank, Ace, King, Queen, Jack, and then numeral values from Ten down to Two. In Omaha Hi-Lo, the lowest possible card for the Lo portion of the game is also the Ace. This thesis abbreviates these cards ranks as `A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, 2`. The

players in this game start with a quantity of chips, called their *stack*, for making and matching bets in the game.

A particular player is designated the *button* and he acts as the card dealer from a standard fifty-two card deck. The button deals four cards to each player, face down, for the players' hands. After this, the player seated directly left of the button must bet an amount, called the *small blind*, and the player to his left must make a larger bet, called the *big blind*. The amounts of these blind bets are predetermined as the table stakes. The two players making these bets are also called the small blind and big blind, respectively. These blind bets form the initial *pot* of chips to be won by the game's winners.

After the players receives their cards, the *preflop* betting stage begins. Players take turns making their game actions, starting with the player left of the big blind, proceeding to the left. On each player's turn, he must decide whether to *fold*, forfeiting the game and any previous chips he has bet, *call*, making a bet to equal the highest previous bet (including the big blind bet) in the round, or *raise*, making a wager to increase the bet level. If a player cannot call the other players' bets because she lacks sufficient chips, the player can still call *all-in*, wagering all her chips into a pot with contributions per player up to her total chip contribution into the pot. Any chips wagered by players above that amount form a *side pot* in which the all-in player does not participate. Should the player decide to raise, he must increase the bet level by at least the value of the big blind bet and the last increase in the bet level. In a no-limit poker game, there is no prescribed upper limit for the raise. A player may therefore raise all-in for all her chips as well. The terms *shove* and *jam* are synonyms for raising or calling all-in, symbolically pushing all her chips into the pot. If all players have taken their turns after the last raise, the dealer moves and organizes the wagered chips in the round to the center of the table. If all but one player fold, the

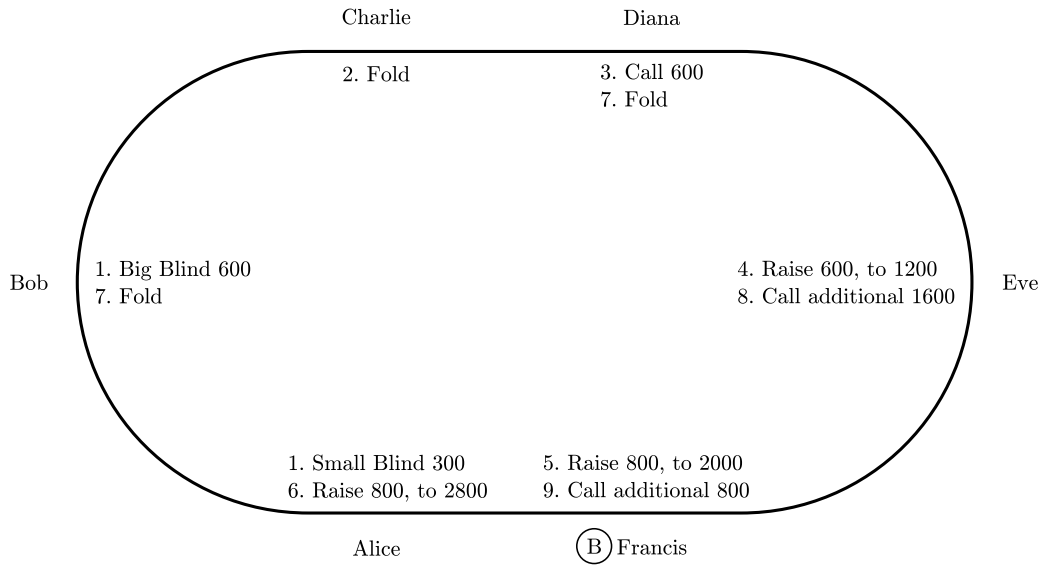

Figure 2.1: Example of a preflop betting round at table stakes of 300/600 for the small and big blind bets. Francis is the button. 1. Alice and Bob make their blind bets of 300 and 600 chips. 2. Charlie folds. 3. Diana calls the big blind. 4. Eve makes the minimum raise, of 600, to 1,200. 5. Francis makes a raise of 800, bringing the total bet to 2,000. 6. Alice's turn again, she makes the minimum raise, same as Francis' raise of 800, to 2,800. 7. Bob and Diana fold. 8. Eve calls Francis' and Alice's raises, topping up her bet to 2,800. 9. Francis calls Alice's raise, topping up to 2,800. Alice being the last raiser, the preflop round concludes after Francis' call. The pot is 9,600 chips: 2,800 from each of Alice, Francis, and Eve, and 600 each from Bob and Diana.

sole remaining player wins all the chips in the pot. Otherwise, the players remaining in the game continue to the next betting stage. Figure 2.1 shows an example of a preflop betting round.

The *flop* betting stage follows the preflop stage. It starts with the dealer dealing three cards, called the *flop*, face up, to the center of the table. The flop cards are the first three of five cards that form the *board*, a set of cards available to all the players to create the best hand. Once again, players take turns taking their game actions of folding, calling, and raising, starting with the small blind. Players may call the "bet" of zero chips in the round if others do not decide to raise; this action

of calling zero chips is also called a *check*. As before, players continue betting and calling until all players respond to the last raise, or all players check.

The *turn* betting stage follows the flop, with the dealer dealing a single card, face up, into the board. Players take turns again, starting with the small blind. After the turn is the *river*, with the dealer dealing a final single card, face up, into the board. Players again take turns with their bets, starting with the small blind, until all players respond to the last raise.

After the river betting stage, all players that have not folded reveal their hands in a *showdown*. The winner of each pot is determined among these remaining players participating in these pots by the following:

- Half the chips in the pot, called the *Hi*, is won by the player with the highest-ranked poker hand that can be formed using exactly two cards from the player's hand and three cards from the board. Table 2.1 lists the different Hi poker hand types and ranks.

- The other half of the pot, called the *Lo*, is won by the player with the lowest-ranked poker hand that can be formed using exactly two cards from the player's hand and three cards from the board, ignoring straights and flushes. Such a Lo hand may not have a hand higher than an Eight, and may not have any duplicate face values such as pairs. Lo hands are ranked by highest face valued card, with lower values beating higher. Two hands with the same five face values are equal in rank. If no player has a qualifying Lo hand, then the winner of the Hi pot also wins the Lo pot.

A player may use two different sets of two cards to form their best Hi and Lo hands, and can win both pots. Winning both Hi and Lo pots is called *scooping* the

Class	Description	Example	Ranking rule	Hands of class
Straight flush	Five cards of the same suit, face values in sequence	8♠ 7♠ 6♠ 5♠ 4♠	Value of the top card in the sequence	10
Four-of-a-kind	Four cards of the same face value	J♠ J♡ J♢ J♣ A♠	Face value of four-of-a-kind, then face value of kicker	156
Full house	Three-of-a-kind and a pair	9♠ 9♡ 9♣ 7♡ 7♢	Face value of three-of-a-kind, then face value of pair	156
Flush	Five cards of the same suit	A♠ J♠ T♠ 9♠ 3♠	Highest-valued card in the hand, then second-highest, down to lowest	1,277
Straight	Five cards with face values in sequence	T♠ 9♢ 8♣ 7♠ 6♡	Value of the top card in the sequence	10
Three-of-a-kind	Three cards of the same face value	3♠ 3♡ 3♣ A♡ 7♣	Face value of the three-of-a-kind, then the higher kicker, then the lower kicker	858
Two pair	Two pairs	A♠ A♢ 8♡ 8♢ 2♠	Higher-ranked pair, then lower-ranked pair, then the rank of the kicker card	858
Pair	Two cards of the same face value	J♠ J♣ A♠ 9♣ 3♡	Face value of pair, then highest kicker, middle-ranked kicker, lowest kicker	2,860
High Card	Any hand not satisfying any of the above	7♠ 5♣ 4♠ 3♢ 2♡	Highest-valued card in the hand, then second-highest, down to lowest	1,277

Table 2.1: Poker hand Hi classes, in descending order. Any hand of a higher class beats hands of lower class. For hands of the same class, the ranking rule determines the higher-ranked hand

pot. If two or more players have hands of the same rank for either pot, the pot is divided evenly among all such players. If a player is the sole winner of one pot and splits the other pot, he is said to have *quartered* the other player, leaving the opponent with one-quarter of the total pot. The game determines winners of the main pot and any side pots separately to ensure that players cannot win more than their fair stake in the overall pot of bets. The game ends with the winners receiving their portions of the pot, adding them to their chip stack. The small blind in this game becomes the button player in the next game, and a new game begins.

2.1.2 Poker Game Concepts

Poker is heavily researched in game theory as a popular game of both chance and hidden information. There is uncertainty about the players' hands, resulting in opportunities to play deceptively. There is also an element of randomness in the five community cards, even after the betting. This section describes a simple card game with betting, then discusses the concepts of expected value, pot odds, and bluffing. It then relates this simple game back to Omaha Hi-Lo poker.

A Simple Jam/Fold Betting Card Game. This two-player game uses the thirteen Spade cards in a standard deck. The cards are ranked in the same order as in Poker, with the Ace having the highest rank. One player is assigned the position of small blind, and makes a blind bet of \$1. The other player is the big blind, making a blind bet of \$2. Each player receives one card. The small blind moves first, deciding either to fold or raise to \$4. If the small blind folds, the big blind wins \$1, the small blind loses \$1, and the game ends. Otherwise, the big blind can choose to either fold or call. If the big blind folds, she loses \$2 to the small blind. If the big blind calls the bet of \$4, the two players reveal their cards and the player with the higher-ranked card wins \$4 from the other.

Expected Value of Playing. It is apparent that if a player receives the Ace, he should play, knowing that his opponent has a lower ranked card. But a player choosing to play only if holding the Ace will lose in the long run, as he will fold too frequently and lose too much to offset the gains of winning with the Ace. The *expected value* of a player's strategy is the average gain or loss experienced by a player that uses the strategy. The expected value of a big blind's strategy that calls only with the Ace, against an indiscriminate small blind player that bets regardless of his card is:

$$EV\,(\text{big blind}) = \$4 \cdot \frac{1}{13} + (-\$2) \cdot \frac{12}{13} = -\$\frac{20}{13}$$

The big blind can improve this by calling with other highly ranked cards. Again, against the small blind player that bets with anything, a strategy with the big blind calling with the Ace or King, folding everything else, expects:

$$EV\,(\text{big blind}) = \$4 \cdot \frac{1}{13} + \left(-\$4 \cdot \frac{1}{12} + \$4 \cdot \frac{11}{12}\right) \cdot \frac{1}{13} + (-\$2) \cdot \frac{11}{13} = -\$\frac{50}{39}$$

The incremental improvement by calling with the King corresponds to winning four chips with probability $\frac{11}{12}$, the chance that the opponent does not have the Ace, and losing four chips with $\frac{1}{12}$ chance. Calling with the King improves the big blind's expected value, and the big blind should call with the King, even with a chance of losing. Expected values can also be calculated given a specific holding for the big blind, such as when she calls while holding the Seven:

$$EV\,(\text{call with Seven}) = \left(-\$4 \cdot \frac{7}{12} + \$4 \cdot \frac{5}{12}\right) = -\$\frac{2}{3}$$

The Big Blind expects to lose $\$\frac{2}{3}$ every time she calls with the Seven. Never-

theless, this is better than the expected (and certain) loss of two chips from folding, so the Big Blind should still call in this case.

Pot Odds. The Big Blind can actually call with hands even lower than a Seven, with the breakeven case being calling with a Five. The expected value of calling with a Five is:

$$EV\,(\text{call with Five}) = \left((-\$4)\cdot\frac{9}{12} + \$4\cdot\frac{3}{12}\right) = -\$2$$

The specific case when the Big Blind holds the Five relates to the size of the pot and the additional wager required. The Big Blind can only win if the Small Blind has one of three cards, while she loses if the Small Blind holds any of the other nine cards. In the long run, the Big Blind loses three times for every win. However, the Big Blind is calling with only an additional $2, for a potential win of $6 already in the pot. The ratio of the value of the pot to potentially win, to the required bet to call, is called the *pot odds*. Higher pot odds mean that the size of the pot is large relative to the wager size. Strategically, when the pot odds are higher than the odds against winning the hand, the player should call. The occasional large pot will sufficiently offset the more frequent but smaller losses in this situation.

Bluffing. The analysis of this game, so far, has focused on the big blind. The analysis from the small blind is more complex, however, because the big blind decides her play after the small blind. This means the small blind must consider the possibility that the big blind folds with weak hands such as a Two or Three. But while neither play can possibly win showing a Two, it does not mean the small blind should automatically fold with it. *Bluffing* is the deceptive bet by a player with a hand that, if the opponent calls, is expected to lose.

As an illustration of the value and implications of bluffing, consider a strategy for the small blind, where he bets with any card with at least a 50% chance of being

higher ranked than his opponent's card. This corresponds to raising with Eights or higher; there are six cards ranked above and six cards ranked below Eight. Reasoning with pot odds, the big blind should call with any Ten or higher. The expected value of this strategy for the small blind is therefore

$$EV\,(\text{small blind}) = (-\$1)\cdot\frac{6}{13}+\frac{2}{13}\cdot\left(\$2\cdot\frac{7}{12}+(-\$4)\cdot\frac{5}{12}\right)+\frac{5}{13}\cdot\left(\$2\cdot\frac{8}{12}\right)=-\$\frac{1}{39}$$

Now consider a bluffing version of this strategy, where the small blind plays the same Eights or higher, but also raises with the Two. He now sometimes plays a hand that he cannot win unless his opponent folds. Adjusting for the small blind's new strategy, the big blind now has sufficient pot odds to play with Nines and above. The expected value of this bluffing small blind strategy is

$$EV\,(\text{bluffer}) = (-\$1)\cdot\frac{5}{13}+\frac{2}{13}\cdot\left(\$2\cdot\frac{6}{12}+(-\$4)\cdot\frac{6}{12}\right)+\frac{6}{13}\cdot\left(\$2\cdot\frac{7}{12}\right)=\$0$$

The small blind improves his overall average by betting when holding the Two, even though the Two can never win if the big blind calls. This effect arises from two sources. First, the big blind still folds hands Eight or below, possibly allowing the small blind's Two to win uncontested. Second, the big blind's Nine loses on average, winning against Eights and Twos but losing to Tens and above. These two effects more than offset the increased losses from being caught bluffing with the Two versus a strong hand.

Back to Omaha Hi-Lo Poker. This game, despite its simplicity, still provides insight into Omaha Hi-Lo Poker. Most importantly, it demonstrates the evaluation of a strategy through its expected value. In Omaha Hi-Lo Poker, while there are many

more than thirteen possible hands, evaluating a strategy still requires calculating the long-run profitability of hands using expected value. It also provides a basis for analyzing the call versus fold decision on the basis of pot odds, and illustrates the value of both raising and calling with hands that will lose more often than win against the opponent.

2.1.3 Properties and Terminology in Omaha Hi-Lo Poker

Omaha Hi-Lo Poker shares many characteristics with other poker games, such as Texas Hold'em poker. The standard poker hand rankings apply to Omaha Hi-Lo, as do the Lo hand rankings from Razz and Lowball poker. But Omaha Hi-Lo, with four cards per player and the requirement that the poker hands be formed with exactly two cards from the player's hand, also has characteristics distinct to this variant.

Four card hands. Each player receives four cards from a fifty-two card deck at the start of the game. This greatly increases the number of possible hand types in Omaha Hi-Lo Poker, compared to Texas Hold'em, where each player receives two cards. In Texas Hold'em, there are $\binom{52}{2} = 1{,}326$ possible hands, while in Omaha Hi-Lo, there are $\binom{52}{4} = 270{,}725$ possible hands. Adjusting for permutations between suits, there are 169 possible hand classes in Texas Hold'em, compared to 16,432 possible hand classes in Omaha Hi-Lo.

Two cards from the hand. The rule that each player uses exactly two cards from their hand creates strategic implications. Compared to Draw Poker, where receiving a four-of-a-kind hand is very strong, it is the worst possible hand to receive in Omaha Hi-Lo. A player receiving four Kings can only rarely do better than creating a hand with two pairs, using two Kings and a pair from the board. It is impossible to create a three-of-a-kind, a straight, flush, or a Lo hand with such a hand. Three-of-a-kind hands are also correspondingly weak.

The two cards rule also impacts the value of holding cards in the same suit. The best possible suit distribution consists of two cards in two different suits, such as A♠ K♠ J♡ T♡. This hand can possibly produce a flush in both spades or hearts. Hands with this suit distribution are called *double-suited*. Hands where there are two or more cards of one suit, such as A♠ 9♠ 3♠ 3♡ are called *single-suited*, and hands where all four cards are of different suit are called *unsuited*. Among the single-suited hands, the best form have two *off-suit* cards, such as K♠ Q♠ 3♡ 2♣, as there are more possible combinations of boards to complement the two suited cards and create a flush. Compare this to the mono-suited hand K♠ Q♠ 3♠ 2♠: while both hands can create a flush when three spade-suited cards appear in the board, there are eleven possible spade cards for the first hand, but only nine in the second.

The four card hand size dramatically increases the straight potential for hands with cards in a tight range. Consider the hand A♠ 9♡ 8♢ 2♣ versus T♠ 9♡ 8♢ 7♣. The only way to create a straight with the first hand is if the board contains at least one of QJT, JT7, T76, or 765. For the second hand, a straight can be formed from many more possibilities: KQJ, QJT, QJ9, QJ8, JT9, JT8, JT7, J98, J97, J87, T96, T86, T76, 986, 976, 975, 965, 876, 865, 765, and 654. *Connectors* are cards with consecutive face values, and hands with many connectors have high straight potential. *Suited connectors* are connectors that also share the same suit, and these also have straight flush potential.

The Lo hand. The split pot structure of Omaha Hi-Lo also gives value to cards from Two to Eight, and additional value to Aces. A hand becomes significantly better if it has two or more Lo cards. A hand with multiple ways to take both Hi and Lo pots has much higher overall expected value than a hand with many high cards. Even A♠ A♢ Q♠ J♢, a double-suited hand with a pair of Aces, has a negative expected value against 5♡ 4♠ 3♣ 2♡, a single-suited hand with four low cards. This effect arises

because the first hand cannot qualify for the Lo pot at all: on almost any board that has three cards between Ace to Eight, the second hand will win half of the overall pot. In many of those cases, the second hand can also form a straight, which beats the pair of Aces. The result is that in this matchup, the first hand has an expected value of 49.47% of the pot, and the second has 50.53% of the pot. The best hand a player can receive in Omaha Hi-Lo combines all of high card strength, low card strength, straight potential, and flush potential: A♠ A♢ 2♠ 3♢.

For a hand to qualify for the Lo pot, the five cards in the hand must all have face values Eight or below, and all values must be different. There are therefore $\binom{8}{5} = 56$ distinct Lo hand types.

2.1.4 Tournament Structure

Poker can be played in two different forms: cash play and tournament play. In cash play, players can enter and exit a poker table between games. To enter a game, a player can bring any number of chips he purchased to make and call the bets in the game. The player can later exit and exchange those chips back into cash. In contrast, in a poker tournament, players all start with the same number of chips, and all start play at the same time. Players may not leave the game except by either losing all their chips, or by winning all of the chips and eliminating all the other players. The players' chips have no value as a result, except as scoring tokens to signify that they remain participants in the tournament. The elimination order of each player determines the ranking of the players in the tournament.

A poker tournament will typically set a low blind bet size at the outset. Players can observe the strategies and playing styles of the other players with only a small risk of elimination. However, the table stakes increase according to a predetermined schedule. This forces the players to play with a wider variety of hands: a player

cannot wait to be dealt strong cards, as the high blind bets whittle away the player's chip stack. A player with 2,000 chips can bide her time for better cards when the blind bets are twenty and forty chips, at 2% of her chip stack; she cannot when the blind bets are 300 and 600 chips, at almost a third of her total stack.

An immediate strategic implication of the rising blinds is that players' strategies shift towards a preflop jam-or-fold model. Suppose a player (Hero) has 2,000 chips, at blinds level of 300/600, and on his turn can choose to either play or fold his hand. The pot, through the blinds, is already at 900 chips. The minimal playing action, calling the big blind of 600, increases the pot to 1,200 chips, while reducing his remaining stack to 1,400 chips. If another player later raises to a bet of 2,000, Hero faces a conundrum. He can call and risk his remaining 1,400 chips to possibly win 3,500 chips, or fold. He is almost always receiving sufficiently high pot odds to call and therefore should call. In effect, Hero calling the big blind commits him to call all further bets and risk all his 2,000 chips. Observing this, he should instead have raised all-in with all 2,000 chips, depriving his opponents of the option to call the 600 chip big blind, with the same 2,000 chips at risk. As such, if Hero decides to not fold, he should *jam* all his chips into the pot.

Tournament poker is a very popular game format, with the online poker site PokerStars hosting thousands of poker tournaments at any given time. A typical PokerStars impromptu "Sit-and-Go" six-player tournament will provide each player with 1,500 chips, and set the starting blinds of 10/20 chips. Blind levels increase every ten minutes. By the time only two players remain, the small and big blinds are frequently as high as 200/400 or 300/600 chips. With the high table stakes, these two players often will play with jam/fold strategies. Assuming that both players will use jam/fold techniques, we compute near-perfect strategies for them in this project.

2.2. Game Theory

The aspect of poker that distinguishes it from other games such as blackjack, lotteries, and other games of chance is the interaction between the players. The players can make strategic decisions and adapt to opponents' strategies, in an effort to win and profit. Other games, such as chess, checkers, and backgammon share this interaction between players, but poker also has an element of hidden information in the players' hands. A rigorous analysis of poker requires a basic understanding of game theory, in particular about concepts such as strategy, regret, and equilibrium.

Strategy. The basic premise in games between players is that the players can choose their actions. The players' individual choices can affect the final outcome. For example, in the game of Rock-Paper-Scissors, a player can choose from one of these three items, which determines whether he wins or loses based on his opponent's action. Define a player's *strategy* σ to be the action the player will take at any game setting where the player's decision can affect the final outcome. These strategies can be *pure*, meaning that the player always chooses the same action at that decision point, or *mixed*, where the player decides among different possible actions according to a probability distribution. In Rock-Paper-Scissors, "always pick rock" is a pure strategy. In an Omaha Hi-Lo Poker Jam/Fold tournament, the only actions available to the small blind are to Raise All-In or Fold. The small blind, for example, may decide to jam all his hands except for unsuited hands, which he jams with probability 0.3. This decision rule, defining the action selected for every possible decision point, is a strategy for the small blind. A *strategy profile* is a set of strategies, one for each player, so that every possible decision point by any player has an associated assigned action.

Best Response. Strategic thinking in a game requires more than blindly choosing among possible actions; players can possibly achieve better results by evaluating and adapting to the opponents' strategies. The *best response* is a strategy σ' with the maximum expected value for a player, given that the opponent played σ. In the Rock-Paper-Scissors example, the best response to "always pick rock" is "always pick paper," winning every time. The best response to "pick either rock or scissors, 50% of the time" is "always pick rock," on average winning every other game and tying the other. The basic form of the big blind's best response is that, based on the small blind's strategy σ^{SB} of what hands to raise and how frequently, to call with hands whose odds of winning are better than the pot odds offered. The best response strategy is also called the *maximally exploitative* strategy, as it takes as much advantage as it can against the opponent.

Regret. Informally, *regret* is the player's observed loss compared to what was possible. Mathematically, define a player P's regret for playing their own strategy σ^P *instead of an alternative strategy* σ_A^P as the difference in expected value for playing σ_A^P and playing σ^P. Positive regrets mean that the alternative strategy performs better than P's own strategy, so P regrets using σ^P instead of σ_A^P. The larger the regret, the larger the opportunity to gain, and therefore the more P regrets not using σ_A^P. Negative regrets can be interpreted as P's "thankfulness" for choosing σ^P instead of σ_A^P.

When the alternative strategy is not mentioned, P's regret for playing σ^P is P's regret compared to the best response possible against the opponent. By definition, a player's regret is non-negative, since a player regrets nothing for using the best possible strategy against the opponent.

Nash Equilibrium. When a player P has a positive regret for using strategy σ^P, it means that there is an alternative strategy σ_A^P better for P. That means a

strategy profile where P uses σ^P is not in equilibrium; P should switch to σ^P_A. A strategy profile where all players have no regrets for using their respective strategies is in equilibrium, called a *Nash Equilibrium*. No player can improve their outcome by choosing a different strategy.

Back to Rock-Paper-Scissors, suppose players A and B follow a strategy profile σ^A = "Play Rock, Scissors, Paper at random, each $\frac{1}{3}$ of the time", σ^B = "Always play Rock." Player B has no regrets, because B can do no better than win, lose, or draw $\frac{1}{3}$ of the time each. But Player A has a positive regret for not choosing "Always pick Paper," so strategy profile $\left(\sigma^A, \sigma^B\right)$ is not a Nash Equilibrium. The strategy where both players use "Play Rock, Scissors, Paper at random, each $\frac{1}{3}$ of the time" is a Nash Equilibrium; neither player can improve against the opponent by changing strategies. There always exists at least one Nash Equilibrium strategy profile in every game (Von Neumann & Morgenstern, 2007; Nash, 1950).

A computer might not be able to compute an exact Nash Equilibrium, but it might find an approximate solution. An *ϵ-Nash Equilibrium* is a strategy profile where no player has regret exceeding a small ϵ; a player might have a better strategy, but the improvement is de minimis.

Minimax. The existence of a Nash Equilibrium does not imply that the equilibrium strategy profile is fair. Games can have a scoring structure that makes the game unfair for one side. In this case, the best a player can do is minimize their loss against the opponent. This amount of loss is called the *minimax* value, or just value, of the game. The name derives from "minimal maximum," as the player seeks to minimize the opponent's score, assuming the opponent will attempt to maximize their own score. The Omaha Hi-Lo Poker Jam/Fold game has non-zero minimax values, dependent on the stakes and the number of chips each player holds.

Regret Matching. The concept of regret lends itself to a strategy learning mechanism through iterative play. People naturally learn to play games better through practice. A beginner player might decide on her actions at random, and observe the results. When she wins, she is thankful for taking her action, and when she loses, she regrets not taking some other action that would have performed better. Over time, she learns which actions she tends to regret making, which actions she would have preferred to take, and which actions she is most thankful for taking. She then shifts her strategy to take the preferable actions more frequently, and the poorer actions less frequently.

A computer can use a similar learning mechanism, acting out all the players in a game. Like the beginner player, the computer starts by modelling each player's strategy as uniformly random. The computer calculates the regret for a player P's model strategy, compared to each of P's pure strategies, assuming that the opponents follow their corresponding model strategies. It also keeps P's running totals of the regrets for each strategy. Afterwards, the computer constructs a new strategy for P, never using any action that P is thankful not to play, and using actions that P wishes it chose, in proportion to the amount P wishes it used that alternative action. The computer repeats this for each player to build a new strategy profile.

The computer runs this process repeatedly, constructing strategy profiles by selecting actions in proportion to the accumulated regret for not using that action. In a game such as poker, where one player's gain is another player's loss, the sequence of strategy profiles that *match the proportion of regrets* over time converge to a Nash Equilibrium (Hart & Mas-Colell, 2000; Robinson, 1951).

2.3. Prior Work in Computer Poker and Poker Strategy

Research into poker has focused on the most popular poker variant, Texas Hold'em. While Omaha Poker and its variants are similar to Texas Hold'em in their use of community cards, Omaha Hi-Lo Poker is more complex to analyze due to its greater number of combinations in hand types and the Hi/Lo split pot structure. With only 169 different hand classes in Texas Hold'em, there are many widely published guidelines for preflop play of different hand types (Sklansky & Malmuth, 1999; Harrington & Robertie, 2010). There are few such systems for Omaha Hi-Lo Poker, with 16,432 different hand classes. Furthermore, published starting hand guidelines for Omaha Hi-Lo Poker are imprecise and qualitative. The Hutchison Point System is one of the few such evaluation systems available for Omaha Poker, though it assumes "a ten-handed game at the lower levels with a mix of good and poor players" (Hutchison, 1997).

This thesis builds upon the work of Miltersen and Sørensen, who published "A Near-Optimal Strategy for a Heads-Up No-Limit Texas Holdem Poker Tournament" (Miltersen & Sørensen, 2007). They found a pair of strategies in two-player jam/fold Texas Hold'em tournament, for their respective small and big blinds, forming an ϵ-Nash Equilibrium. In doing so, Miltersen and Sørensen demonstrated the feasibility of computing such an equilibrium in a game where each player has one of 169 possible hand classes, using linear optimization. A year later, Ganzfried and Sandholm published a similar equilibrium for three players (Ganzfried & Sandholm, 2008).

This thesis draws inspiration from the SAGE System, an approximation of a Nash Equilibrium for a Texas Hold'em tournament end-game situation (Jones, 2006). The SAGE System is a set of scoring rules for the cards in a Texas Hold'em poker hand, with which a player can quickly calculate a Power Index score. The player then

compares the computed Power Index to a threshold score; if the hand's Power Index exceeds the threshold, the player should jam all-in.

Equilibrium results in Texas Hold'em research have used simplifying assumptions to shrink the search space. In particular, the SAGE System, the two player ϵ-Nash Equilibrium and the three player ϵ-Nash Equilibrium all restricted their analysis to jam/fold situations. This is a common assumption, as described in Sklansky's "The System" (Sklansky, 2007). The assumption is not necessary, however, as demonstrated in Cepheus, an approximate equilibrium strategy for two-player Limit Texas Hold'em Poker.

The University of Alberta's Poker Research Group developed Cepheus through a machine learning algorithm called CFR+ (Bowling et al., 2015; Tammelin, 2014). This algorithm is a new refinement of the Counterfactual Regret Minimization (CFR) algorithm developed by the University of Alberta (Zinkevich et al., 2007), in turn a successor to regret matching. CFR is well-suited to games with many different potential decision points, scaling linearly to the number of such points, instead of scaling exponentially using regret matching or linear optimization. Cepheus used nearly a one thousand CPU-years of computation to learn an unexploitable strategy for every possible contingency.

In this project, we calculate an ϵ-Nash Equilibrium for a two-player jam/fold Omaha Hi-Lo tournament game. This project uses the CFR+ algorithm to compute the equilibrium, as CFR+ is better suited for games with many possible game states than linear optimization. Finally, it finds a hand scoring system for Omaha Hi-Lo tournament endgames, similar to the SAGE System for Texas Hold'em tournament endgames. While this project does not require even a CPU century of computer time, it still requires extensive computation; Amazon Web Services (AWS) and the OpenCL framework make powerful computers and specialized processors available

inexpensively.

2.4. Amazon Web Services

AWS is the Cloud Computing division of Amazon.com. It provides web services and computing resources to users that require computing capacity flexibly without upfront hardware costs. Instead, AWS charges users by hours of server usage, gigabyte-months of storage, and gigabytes of network transfer. AWS offers the Elastic Compute Cloud (EC2) service for turnkey computer servers. Amazon offers these servers in three pricing models: on-demand, spot, and reserved instances. On-demand instances run continuously, with fees accruing per hour according to Amazon's published price schedule. Spot instances run based on auction pricing: if the user's offered price for an instance exceeds the spot market rate, the server starts, and if the market rate later exceeds the offered price, AWS shuts the server down. Reserved instances require the user to pay an upfront or annual commitment fee, with a corresponding discounted per hour fee, with assured capacity. Spot market prices for an instance can be deeply discounted compared to on-demand prices: a server with a quad-core Intel processor and sixteen gigabytes of memory can cost as little as 2.65 cents per hour in an `m4.xlarge` spot pricing instance, compared to 25.20 cents per hour on-demand. This thesis project used AWS EC2, in on-demand and spot pricing models, and Simple Storage Service (S3) for large bursts of computing power and storage.

2.5. OpenCL

OpenCL is an industry standard computing framework for writing programs that execute in a heterogeneous computing environment. Programs written using the OpenCL framework can execute on central processing units (CPUs), graphics pro-

cessing units (GPUs), and other processors. Similar to CUDA by NVIDIA, OpenCL is commonly used to offload highly parallel computing tasks to GPUs, performing general purpose computing on graphics processing units (GPGPU). Programs using GPGPU can be over 100 times faster than CPU-only processes (Stone et al., 2010). OpenCL can also be used to spawn parallel processes on CPU-only computers. The OpenCL platform remains current, with version 2.1 ratified in March 2015.

AWS provides access to servers with dedicated GPUs with support for OpenCL in their `g2.2xlarge` and `g2.8xlarge` instance types. The servers available through EC2 also all support OpenCL without GPUs using CPU processing.

Chapter 3: Methodology and Design

This chapter will describe the stages taken to construct the heuristic scoring system that is the goal of this project. The first section provides a high-level overview of the Hand Evaluator, Game Tabulator, Matchup Tabulator, Equilibrium Solver, and Feature Valuation components and the system architecture that integrates them. The following sections then describe these components in detail.

3.1. Overview

This project required the construction of five distinct components: 1. a fast Hand Evaluator to determine the winner of a showdown between two players, 2. a Game Tabulator that compares two four-card hands in a matchup and reports the distribution of win/quarter/tie/loss results, 3. a Matchup Tabulator that records these results for all possible matchups, 4. an Equilibrium Solver to learn a ϵ-Nash Equilibrium strategy for the two players, and 5. Feature Valuation of a four-card hand to compute a Power Level score of that hand.

Both cards and hands required a consistent enumeration method throughout this project. For simplicity in separating out the suits and face values of a given card, this project uses the card numbering convention `A♣ = 0, 2♣ = 1, ..., K♣ = 12, A♢ = 13, ..., A♡ = 26, ..., K♠ = 51`.

The numbering of an n-card combination is given by its combinatorial index

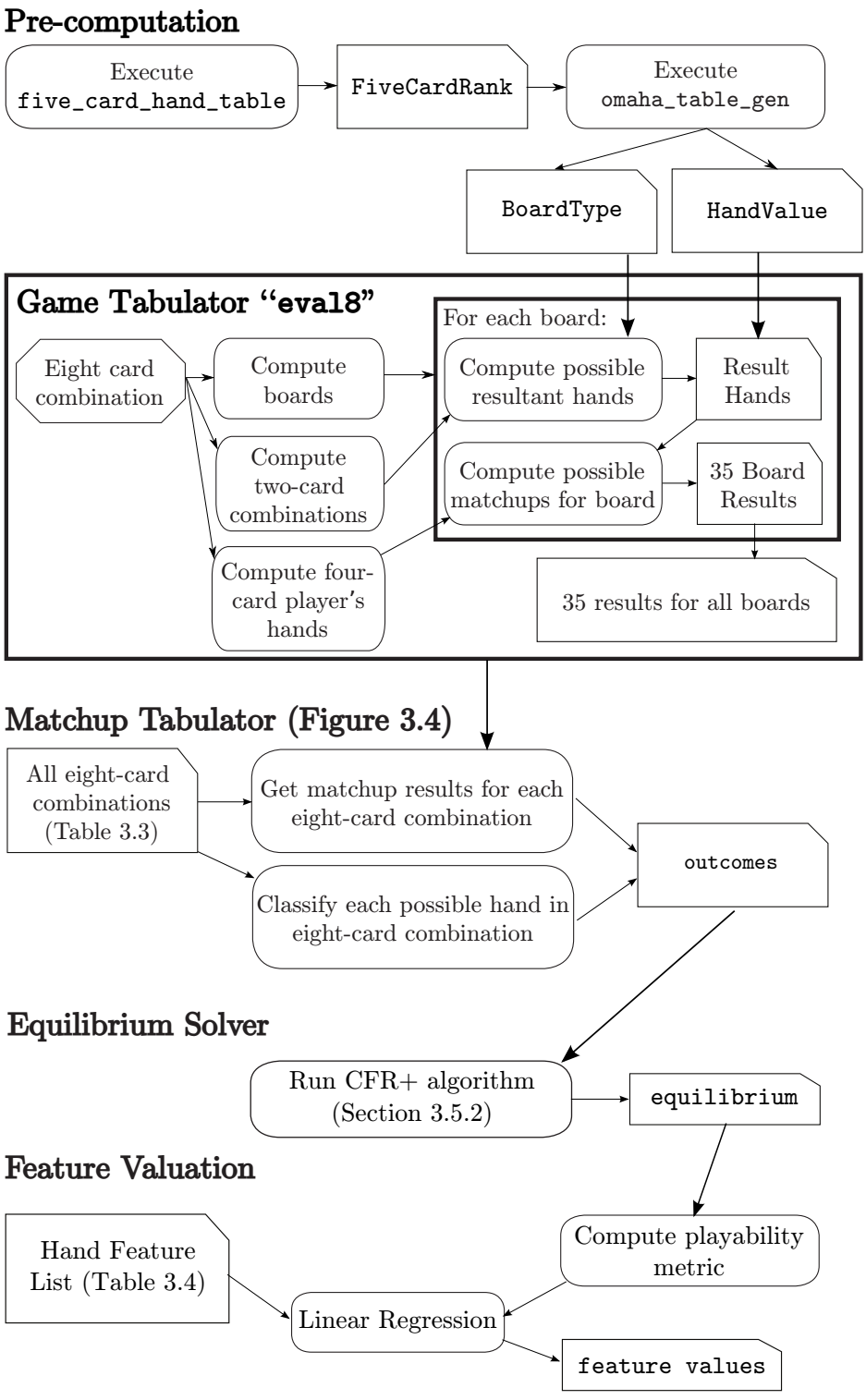

Figure 3.1: Logical flow between Project Components

value,

$$N(c_1, c_2, \ldots, c_n) = \sum_{i=1}^{n} \binom{c_i}{i}$$

where the cards c_i are ordered so that $c_1 < c_2 < \ldots, c_n$ and $\binom{n}{m} = 0$ if $m > n$. For example, the four-card combination K♠ J♡ T♠ 6♢ corresponds to $(c_1, c_2, c_3, c_4) = (18, 36, 48, 51)$ and its index is $\binom{51}{4} + \binom{48}{3} + \binom{36}{2} + \binom{18}{1} = 267{,}844$. The four-card combination with the lowest index is A♣ 2♣ 3♣ 4♣, corresponding to $(c_1, c_2, c_3, c_4) = (0, 1, 2, 3)$ and index $\binom{3}{4} + \binom{2}{3} + \binom{1}{2} + \binom{0}{1} = 0$. The four-card combination with the highest index is K♠ Q♠ J♠ T♠, with an index of $\binom{51}{4} + \binom{50}{3} + \binom{49}{2} + \binom{48}{1}$ = 270,724. All $\binom{52}{4}$ = 270,725 four-card combinations have a unique index between 0 and 270,724.

This thesis uses the phrase "maps an n-card combination to a class" to mean "calculate the index value of the n-card combination and retrieve the value in a lookup table using that index value as the search key."

3.2. Hand Evaluator

In poker, the player revealing the highest ranked hand is the winner of the showdown. Determining the winner of a showdown, therefore, requires answering the fundamental question for each player's hand: "what is the rank of this hand?" The Hand Evaluator answers this question.

The construction of a fast Hand Evaluator in this project required computing three preliminary lookup tables. The first lookup table $FiveCardRank : \left(0 \ldots \binom{52}{5}\right) \rightarrow \texttt{int}$ maps a five-card combination to the Hi and Lo poker hand ranks for these five cards. The second lookup table $BoardType : \left(0 \ldots \binom{52}{5}\right) \rightarrow \texttt{int}$ maps a five-card combination to a community card board class, and the third lookup table $HandValue : (0 \ldots 152619, 0 \ldots 402) \rightarrow \texttt{int}$ maps a board type and two-card hand

type index to a final hand rank value.

The `five_card_hand_table` program computes all the values in the *FiveCardRank* table. It encodes the Hi and Lo hand ranks into a thirty-two bit value, using four bits for a Hi poker hand class, twelve bits for the rank of that hand within that type, and eight bits for the Lo hand rank.

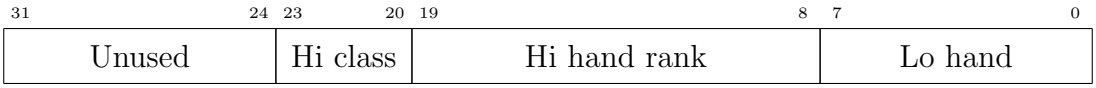

Figure 3.2: Encoding of Hi and Lo hand ranks

The eight Lo hand bits encode either zero, corresponding to no Lo hand, or a number from one to fifty-six, corresponding to one of the fifty-six possible Lo hand classes, with the value of fifty-six representing the best possible Lo hand, `5432A`. The four Hi hand class bits encode a number from one (High Card) to nine (Straight flush) and the twelve Hi hand rank bits encode a number from zero to 2,859. These sixteen bits distinguish each of the hand classes and ranks in Table 2.1. Figure 3.2 illustrates the bit assignments for these ranks.

This encoding allows for a fast comparison of poker hand ranks: for two hands H_1 and H_2, determining whether H_1 beats H_2 in the Hi hand can be done by looking up $FiveCardRank(H_1)$ and $FiveCardRank(H_2)$, dropping the eight Lo bits of each, and evaluating whether $FiveCardRank(H_1)$ `& 0xffffff00` $>$ $FiveCardRank(H_2)$ `& 0xffffff00`. In turn, determining the winner of the Lo hand can be done by comparing the Lo bits of the two *FiveCardRank*s.

The `omaha_table_gen` program uses the *FiveCardRank* table to compute the *BoardType* and *HandValue* lookup tables. To construct the *BoardType* lookup table, the `omaha_table_gen` program iterates through the $\binom{52}{5}$ possible five-card combinations, decomposing each combination into a same-suit classification and a face

value classification. Because in Omaha Poker the player must use exactly three of the five community cards to construct their final hands, a combination of five cards without three cards of the same suit precludes any flush or straight flush hand. The program evaluates the same-suit classification of five cards as one of: five cards of the same suit, four cards of the same suit and one card of a different suit, three cards of the same suit and two cards of a suit different from that (but not necessarily different from each other) and no three cards of the same suit. In conjunction with this, the program classifies face values, retaining the suit information of three or more cards of the same suit. Each five-card combination can thus be classified into one of 152,607 types:

Five cards of the same suit

There are $\binom{13}{5} = 1{,}287$ possible combinations of five cards, all of the same suit, none with the same face value, of thirteen possible face values.

Four cards of one suit and one card of another suit

There are $\binom{13}{4} = 715$ possible combinations of four cards of the same suit, and thirteen possible face values for the final card (the card's suit is irrelevant), totalling 9,295 classes of this type.

Three cards of one suit and two card of other suits

There are $\binom{13}{3} = 286$ possible combinations of three cards of the same suit. There are $\binom{14}{2} = 91$ possible combinations of face values for the other two cards, since the face values of these two cards can repeat, such as in the example `Ks Qs Js Kd Kh`. There are 26,026 classes of this type.

No three cards of the same suit

There are $\binom{17}{5} = 6{,}188$ possible combinations of five cards from thirteen possible

face values, with face values selected with replacement. However, since there are only four cards of each face value total, the thirteen combinations with five cards all with the same face value never occur. There are 6,175 boards of this type. For convenience in recalculating the index of these boards, the impossible combinations are still assigned an index value.

For the first three types, there are four possible relevant suits. There are thus 1,287 + 715 + (26,026 × 4) + 6,188 = 152,620 types of boards in total. The `omaha_table_gen` program assigns indices for each type as described in Table 3.1.

	Index	Description of board
Clubs	0 - 1,286	Five clubs
	1,287 - 10,581	Four clubs
	10,582 - 36,607	Three clubs
Diamonds	36,608 - 37,894	Five diamonds
	37,895 - 47,189	Four diamonds
	47,190 - 73,215	Three diamonds
Hearts	73,216 - 74,502	Five hearts
	74,503 - 83,797	Four hearts
	83,798 - 109,823	Three hearts
Spades	109,824 - 111,110	Five spades
	111,111 - 120,405	Four spades
	120,406 - 146,431	Three spades
Unsuited	146,432 - 152,619	Fewer than three cards per suit

Table 3.1: *BoardType* **table layout**

Similar to the *BoardType* function that maps a five-card combination to a community card class, a *TwoCardType* mapping function classifies the $\binom{52}{2}$ possible two-card combinations into either two cards of the same suit, or two cards of a different suit. There are $\binom{13}{2} = 78$ possible two-card combination types where the two cards have different face value, and 13 types where the two cards have the same face value.

Each combination of two cards with different face values can be further classified as one of: two clubs, two diamonds, two hearts, two spades, or two cards of different suit. There are a total of $78 \times 5 + 13 = 403$ total two-card combination types.

To construct the $HandValue$ lookup table, the `omaha_table_gen` program iterates through all five-card combinations of community cards, along with all two-card combinations of possible player cards. With each such combination, the program looks up the ten possible poker hands using both player cards and exactly three community cards, and tracks the highest Hi and Lo hand values. The program then evaluates the $BoardType$ and $TwoCardType$ values of the five community cards and two player cards, encodes the best Hi and Lo ranks as $BestRank$ in the same method as in the $FiveCardRank$ lookup table, and stores $HandValue(BoardType, TwoCardType) = BestRank$ in the lookup table. Table 3.2 describes the layout of the $HandValue$ lookup table.

		Four-card Hand			
		0	1	...	402
Five-card Board Class	0	Best Hand: Hi and Lo Ranks			
	1				
	2				
	...				
	152619				

Table 3.2: $HandValue$ lookup table, combining Hand and Board classes for Hi/Lo hand ranks

With the $BoardType$, and $HandValue$ lookup tables created by `omaha_table_gen`, a Hand Evaluator can now answer the question "what is the rank of this hand?" given the five community cards and four player cards:

- Use the $BoardType$ lookup table to determine the board type of the five community cards,

- Compute the six $TwoCardType$ values from the possible two-card combinations in the player's four cards,
- Use the $HandValue$ lookup table for the determined $BoardType$ class and $TwoCardType$ classes to determine the six possible Hi and Lo hands,
- Get the rank of the highest ranked Hi and Lo hands from these six hands.

3.3. Game Tabulator

Determining the winner of a showdown is an important aspect of poker, but it is a backward-looking activity. There can only be a winner of a showdown at the end of the game, after all the players have made their decisions. As such, the Hand Evaluator does not help the player to answer the question "should I play or should I fold this hand?" A Game Tabulator provides insight into this question by answering the question "what do I expect to win with this hand, if I knew my opponent had that hand?"

As a baseline, evaluating a single showdown given two four-card hands and five community cards requires six $TwoCardType$ evaluations for each of the two hands, one $BoardType$ lookup for the five community cards, and six $HandValue$ lookups per player to determine that player's overall Hi and Lo hand ranks. With $\frac{\binom{52}{4}\binom{48}{4}}{2} = 26{,}338{,}835{,}250$ possible preflop matchups and $\binom{44}{5} = 1{,}086{,}008$ possible showdowns per preflop matchup, a brute-force evaluation of every showdown would require approximately 3.43×10^{17} iterations of $HandValue$ lookups. Even at one billion showdown evaluations per second, calculating all these results would require almost one year.

We reduce the number of evaluations required by exploiting three symmetries in the game. The first symmetry is across community card combinations. Observe

that dealing the two players' hands can be split into two steps: pre-selecting eight player cards from the deck, then distributing four cards to each player from these eight selected cards. On selecting the eight player cards, the remaining forty-four cards in the deck becomes fixed. This also fixes the possible community card boards, independent of the actual hands the players receive. The second symmetry is across two-card combinations. The six possible two-card combinations that each player can use to create their Hi and Lo hands will be among the $\binom{8}{2} = 28$ two-card combinations from the eight selected player cards. The third symmetry is across community card combination classes. Each of the $\binom{44}{5} = 1{,}086{,}008$ possible community card combinations can still be classified as a *BoardType* from Table 3.1.

The `eval8` algorithm exploits these three symmetries to efficiently determine the distribution of matchup outcomes. It takes an input of the eight player cards. `eval8` starts by determining the frequency of each *BoardType* among the 1,086,008 possible boards. Then, for each *BoardType*, `eval8` retrieves the Hi and Lo *HandValues* for the twenty-eight possible *TwoCardTypes* and stores the outcomes in a small lookup table. `eval8` then constructs the $\binom{8}{4} = 70$ possible hands using the eight player cards, and determines the best resulting hands using this table. Next, `eval8` determines the winner in each of the thirty-five possible matchups based on the values of the seventy hands. The `eval8` algorithm concludes after evaluating all the possible boards by summing the results for each board, weighted by the frequency of each *BoardType*. Figure 3.3 illustrates the use of pre-evaluated *HandValue* results for two possible matchups given an input of eight player cards. Exploiting these three symmetries reduces the number of *HandValue* lookups for thirty-five matchups from the brute-force computation with $35 \times 1{,}086{,}008 \times 12 = 456{,}123{,}360$ lookups to $152{,}620 \times 28 = 4{,}273{,}360$, for a reduction of over 99%.

The `GameTabulator` program implements the `eval8` algorithm to compute

Card 1	Card 2	Hi	Lo
3♣	8♣	♣ Flush, AQ873	8763A
3♣	9♣	♣ Flush, AQ973	None
3♣	2♢	77A32	7632A
3♣	T♡	77AT3	None
3♣	A♠	AA773	None
3♣	5♠	77A53	7653A
3♣	J♠	77AJ3	None
8♣	9♣	♣ Flush, AQ987	None
8♣	2♢	77A82	8762A
8♣	T♡	77AT8	None
8♣	A♠	AA778	None
8♣	5♠	77A85	8765A
8♣	J♠	77AJ8	None
9♣	2♢	77A92	None
9♣	T♡	77AT9	None
9♣	A♠	AA779	None
9♣	5♠	77A95	None
9♣	J♠	77AJ9	None
2♢	T♡	77AT2	None
2♢	A♠	AA772	None
2♢	5♠	77A52	7652A
2♢	J♠	77AJ2	None
T♡	A♠	AA77T	None
T♡	5♠	77AT5	None
T♡	J♠	77AJT	None
A♠	5♠	AA775	None
A♠	J♠	AA77J	None
5♠	J♠	77AJ5	None

Figure 3.3: `eval8` **process for eight-card combination of** 3♣ 8♣ 9♣ 2♢ T♡ A♠ 5♠ J♠ **and board of** A♣ 7♣ Q♣ 6♢ 7♢**. 1. Evaluate all twenty-eight possible hands from board and two-card combinations from hand. 2. Look up the six possible two-card hands from each four-card hand. 3. Determine winner of Hi and Lo hands.**

the number of scoops, draws, quarters, and ties for the thirty-five matchups arising from an eight-card player card combination. In this implementation, `GameTabulator` uses the OpenCL framework to offload the $HandValue$ lookups to a graphics accelerator. A graphics accelerator, tuned for highly parallel processing, can evaluate thirty-two different $BoardTypes$ simultaneously. The OpenCL framework allows `GameTabulator` to dispatch table lookups and evaluate the Hi and Lo hands using the results of those lookups as a workgroup task, and even run workgroups in parallel. Running the workgroups in parallel permits the graphics processor to calculate Hi and Lo hands while other workgroups incur memory latency when waiting for $HandValue$ table lookups. The performance gain using GPU processing is noticeable: `GameTabulator` completes 500 runs of `eval8` in fifteen seconds using the graphics accelerator on an Amazon `g2.2xlarge` instance, compared to twenty seconds on the eight-core CPU of the same instance, reducing the computation time by 20%.

3.4. Matchup Tabulator

The limitation of `GameTabulator` lies in the input assumption that it knows the opponent's exact hand. The player in the small blind position has no information about his opponent's hand at all, and yet he must make his decision to raise or fold his hand. This player must run the Hand Evaluator for every possible opponent hand, and calculate the frequency of each possible opponent hand, to compensate for the opponent's hidden holding. The `MatchupTabulator` runs `GameTabulator` to evaluate every possible eight-card player card combination.

There are $\binom{52}{8} = 752{,}538{,}150$ eight-card player card combinations that `MatchupTabulator` must evaluate. However, observe that permuting the suits in a matchup between two four-card hands does not affect the distribution of outcomes.

For example, the hand A♠ J♢ 9♠ 9♢ will perform equally against K♠ Q♠ 3♢ 2♡ as A♡ J♣ 9♡ 9♣ against K♡ Q♡ 3♣ 2♠. The possible eight-card combinations of fifty-two cards can therefore be classified into equivalent combination types, letting `MatchupTabulator` evaluate only one item of each class, using symmetry to infer the results of the equivalent combination types. There are 32,819,436 such types, described in Table 3.3.

`MatchupTabulator` only needs to evaluate one eight-card combination from each of these types, shrinking the search space by 95.5% compared to evaluating every possible eight-card combination. The `GenerateTableClasses.py` program computes representative combinations for each eight-card combination class, creating a comma-separated values file of class index and eight cards.

`MatchupTabulator` coordinates multiple processing nodes running the `GameTabulator` program to process the eight-card classes more quickly. It uses a simple SQLite database to track previously evaluated combinations, and send work packets to the individual processing nodes. The `GameTabulator` program returns, for each eight-card combination in its work packet, the results of the thirty-five matchups, as well as the two players' hand classes in each matchup. The `MatchupTabulator` server validates that the total number of results is 1,086,008 outcomes per match, each eight-card combination has thirty-five matchups, and that the processing node evaluated all the eight-card combinations in its work packet. After `MatchupTabulator` validates the response, it uploads the results as comma-separated values files to S3 for storage. The `MatchupTabulator` consists of two web-server CGI scripts, `GetMatchups.py` and `ReceiveResults.py`, which distribute and receive work packets and results. The `EvaluatorWorkerNode.py` client retrieves work units from the web server and runs `GameTabulator` to compute hand classes and matchup results.

Suit distribution and characteristics	Example	Code	Classes	Duplicates	Total
(8, 0, 0, 0)	A♠ K♠ Q♠ J♠ T♠ 9♠ 8♠ 7♠	C_A8B0C0D0	$\binom{13}{8}$	4	5,148
(7, 1, 0, 0)	A♠ K♠ Q♠ J♠ T♠ 9♠ 8♠ A♡	C_A7B1C0D0	$\binom{13}{7}\binom{13}{1}$	12	267,696
(6, 2, 0, 0)	A♠ K♠ Q♠ J♠ T♠ 9♠ A♡ T♡	C_A6B2C0D0	$\binom{13}{6}\binom{13}{2}$	12	1,606,176
(6, 1, 1, 0), pair in unsuited cards	A♠ K♠ Q♠ J♠ T♠ 9♠ A♡ A♢	C_A6B1C1D0m0	$\binom{13}{6}\binom{13}{1}$	12	267,696
(6, 1, 1, 0), not pair in unsuited cards	A♠ K♠ Q♠ J♠ T♠ 9♠ A♡ 2♢	C_A6B1C1D0m1	$\binom{13}{6}\binom{13}{2}$	24	3,212,352
(5, 3, 0, 0)	A♠ K♠ Q♠ J♠ T♠ A♡ 8♡ 6♡	C_A5B3C0D0	$\binom{13}{5}\binom{13}{3}$	12	4,416,984
(5, 2, 1, 0)	A♠ K♠ Q♠ J♠ T♠ A♡ 8♡ 3♢	C_A5B2C1D0	$\binom{13}{5}\binom{13}{2}\binom{13}{1}$	24	31,320,432
(5, 1, 1, 1), pair in unsuited cards	A♠ K♠ Q♠ J♠ T♠ A♡ 8♢ 8♣	C_A5B1C1D1m1	$\binom{13}{5}\binom{13}{2}\binom{13}{1}$	12	2,409,264
(5, 1, 1, 1), trips in unsuited cards	A♠ K♠ Q♠ J♠ T♠ 8♡ 8♢ 8♣	C_A5B1C1D1m2	$\binom{13}{5}\binom{13}{1}$	4	66,924
(5, 1, 1, 1), no pairs or trips in unsuited cards	A♠ K♠ Q♠ J♠ T♠ A♡ 8♢ 3♣	C_A5B1C1D1m0	$\binom{13}{5}\binom{13}{3}$	24	8,833,968
(4, 4, 0, 0), matching four-flush	A♠ K♠ Q♠ J♠ A♡ K♡ Q♡ J♡	C_A4B4C0D0m1	$\binom{13}{4}$	6	4,290
(4, 4, 0, 0), not matching four-flush	A♠ K♠ Q♠ J♠ A♡ K♡ Q♡ T♡	C_A4B4C0D0m0	$\binom{\binom{13}{4}}{2}$	12	3,063,060
(4, 3, 1, 0)	A♠ K♠ Q♠ J♠ A♡ 8♡ 6♡ 3♣	C_A4B3C1D0	$\binom{13}{4}\binom{13}{3}\binom{13}{1}$	24	63,800,880
(4, 2, 2, 0), matching two-flush	A♠ K♠ Q♠ J♠ T♡ 8♡ T♢ 8♢	C_A4B2C2D0m1	$\binom{13}{4}\binom{13}{2}$	12	669,240
(4, 2, 2, 0), not matching two-flush	A♠ K♠ Q♠ J♠ T♡ 8♡ T♢ 6♢	C_A4B2C2D0m0	$\binom{13}{4}\binom{\binom{13}{2}}{2}$	24	51,531,480
(4, 2, 1, 1), unmatched unsuited cards	A♠ K♠ Q♠ J♠ T♡ 9♡ 3♢ 2♣	C_A4B2C1D1m0	$\binom{13}{4}\binom{13}{2}\binom{13}{2}$	24	104,401,440
(4, 2, 1, 1), unsuited cards pair	A♠ K♠ Q♠ J♠ T♡ 9♡ 3♢ 2♣	C_A4B2C1D1m1	$\binom{13}{4}\binom{13}{2}\binom{13}{1}$	12	8,700,120
(3, 3, 2, 0), matching three-flush	A♠ K♠ Q♠ A♡ K♡ Q♡ Q♢ T♢	C_A3B3C2D0m1	$\binom{13}{3}\binom{13}{2}$	12	267,696
(3, 3, 2, 0), not matching three-flush	A♠ K♠ Q♠ A♡ K♡ J♡ Q♢ T♢	C_A3B3C2D0m0	$\binom{\binom{13}{2}}{2}\binom{13}{2}$	24	76,293,360
(3, 3, 1, 1), matching three-flush, pair in unsuited cards	A♠ K♠ Q♠ A♡ K♡ Q♡ J♢ J♣	C_A3B3C1D1mb	$\binom{13}{3}\binom{13}{1}$	6	22,308
(3, 3, 1, 1), matching three-flush, not pair in unsuited cards	A♠ K♠ Q♠ A♡ K♡ Q♡ J♢ T♣	C_A3B3C1D1m3	$\binom{13}{3}\binom{13}{2}$	12	267,696
(3, 3, 1, 1), not matching three-flush, pair in unsuited cards	A♠ K♠ Q♠ A♡ K♡ T♡ J♢ J♣	C_A3B3C1D1m1	$\binom{\binom{13}{2}}{2}\binom{13}{1}$	12	6,357,780
(3, 3, 1, 1), none of the above	A♠ K♠ Q♠ A♡ K♡ T♡ J♢ T♣	C_A3B3C1D1m0	$\binom{\binom{13}{2}}{2}\binom{13}{2}$	24	76,293,360
(3, 2, 2, 1), matching two-flush	A♠ K♠ Q♠ J♡ T♡ J♢ T♢ 9♣	C_A3B2C2D1m1	$\binom{13}{3}\binom{13}{2}\binom{13}{1}$	12	3,480,048
(3, 2, 2, 1), not matching two-flush	A♠ K♠ Q♠ J♡ T♡ J♢ 9♢ 9♣	C_A3B2C2D1m0	$\binom{13}{3}\binom{\binom{13}{2}}{2}\binom{13}{1}$	24	267,963,696
(2, 2, 2, 2), all two-flushes match	A♠ K♠ A♡ K♡ A♢ K♢ A♣ K♣	C_A2B2C2D2m4	$\binom{13}{2}$	1	78
(2, 2, 2, 2), three two-flushes match	A♠ K♠ A♡ K♡ A♢ K♢ K♣ Q♣	C_A2B2C2D2m3	$\binom{13}{2}\left(\binom{13}{2}-1\right)$	4	24,024
(2, 2, 2, 2), two two-flushes match, two other two-flushes match	A♠ K♠ A♡ K♡ A♢ Q♢ A♣ Q♣	C_A2B2C2D2m2p	$\binom{\binom{13}{2}}{2}$	6	18,018
(2, 2, 2, 2), two two-flushes match	A♠ K♠ A♡ K♡ A♢ Q♢ Q♣ J♣	C_A2B2C2D2m1	$\binom{13}{2}\binom{\binom{13}{2}-1}{2}$	12	2,738,736
(2, 2, 2, 2), no matching two-flushes	A♠ K♠ K♡ Q♡ A♢ Q♢ Q♣ J♣	C_A2B2C2D2m0	$\binom{\binom{13}{2}}{4}$	24	34,234,200
Total			32,819,436		752,538,150

Table 3.3: Eight-card combinations and `GenerateTableClasses.py` class codes

Both `MatchupTabulator` server scripts and `EvaluatorWorkerNode.py` clients ran in the AWS environment. The server ran in an AWS `t2.medium` on-demand instance, equivalent to 20% CPU time on a dual-core server with four gigabytes of memory. In contrast, the worker nodes ran in `g2.2xlarge` spot instances, each having a GPU with four gigabytes of memory. The server distributed work packets as requested from the worker. The server also linked each work packet to the assigned node to prevent duplication of work across nodes. Each work packet contained 10,000 eight-card combinations, for an estimated processing time of six minutes per packet. Every hour, the server unlinked work packets that remained incomplete after two hours since the request, on the assumption that the worker node was shut down by AWS or otherwise failed. This is a simple fault-tolerance mechanism, enabling the use of cheaper but potentially volatile spot instances which can be shut down and reallocated by AWS. Figure 3.4 illustrates the process flow between the `MatchupTabulator` server that tracks the work assignments for workers, the Worker Nodes that run the `GameTabulator`, and the S3 storage to archive the workers' results.

After `MatchupTabulator` processes all 32,819,436 eight-card combination classes, the `BuildOutcomes` program takes all the uploaded matchup outcomes and eight-card combination list to create a $(16{,}432 \times 16{,}432) \rightarrow$ `(int, int, int, int, int)` mapping. This *outcomes* table takes two hand classes, and returns the number of scoops, quarters, and ties for all possible matchups between those two hand classes.

3.5. Equilibrium Solver

3.5.1 The Summary Table and Expected Value calculations

`MatchupTabulator` provides the data necessary to evaluate strategies for each player: a complete table of every possible Omaha hand type versus hand type

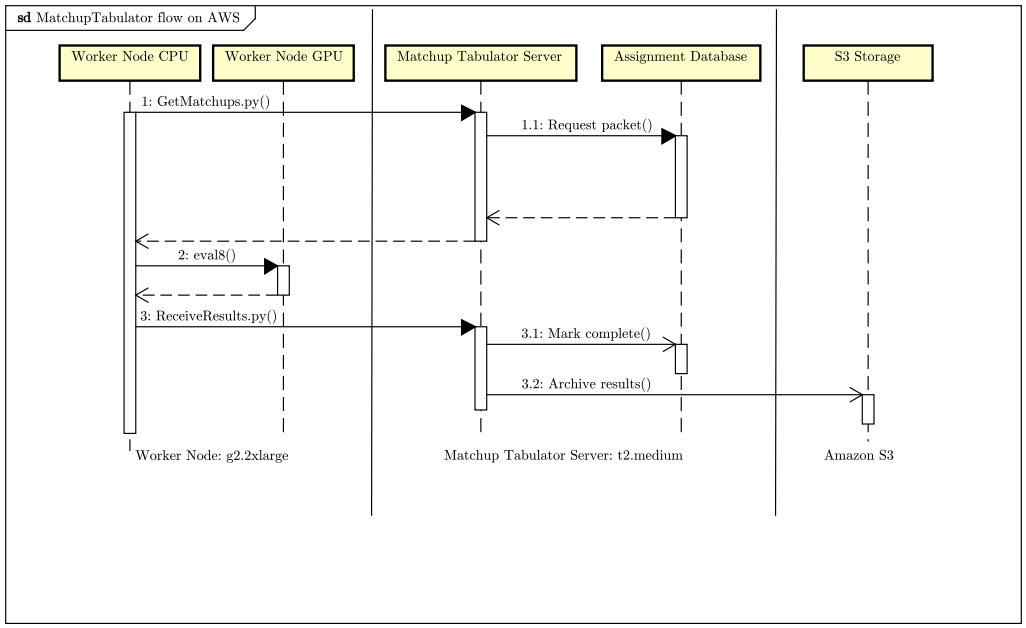

Figure 3.4: Processing sequence between Worker Nodes and Matchup Tabulator Server in AWS

matchup, a table $freq$ containing the frequencies of all possible matchups, and the total distribution of *outcomes* for each matchup. The `solver` program described in this section implements an Equilibrium Solver that can evaluate different player strategy profiles and learn stronger strategies. In doing so, `solver` can answer the player's fundamental question, "should I play or should I fold this hand?"

The *outcomes* table computed by `MatchupTabulator` is a (16,432 × 16,432 × 5) array of 32-bit integers, requiring over five gigabytes of memory. This array, while smaller than the available memory in a modern desktop computer, cannot be loaded into the four gigabytes of memory available in an Amazon `g2.2xlarge` GPU. This project makes two assumptions to pre-process the large table into a smaller data representation that `solver` can fit inside four gigabytes of memory. The Independent Chip Model (ICM) assumption states that a player's chance to win a tournament is equal to their share of the total chips. The rational risk-neutral player assumption

states that a player always prefers options with higher expected utility over lower expected utility, and is indifferent to options with equal expected utility, regardless of any randomness in the options.

For a two-player tournament, the ICM assumption means that a player's equity in an all-or-nothing tournament is equal to his share of the total chips. As a result, a player winning n chips increases his equity by $\frac{n}{\text{total number of chips}}$. This implies that every chip has equal value in a two-player game. The rational risk-neutral player assumption means that the players will raise or call if the unweighted expected value of raising or calling exceeds the expected value of folding.

The expected value from folding is the loss of the players' blinds. We can also calculate the expected value from raising or calling in a matchup using the outcomes distribution calculated by `MatchupTabulator`. Once the two players' hands are dealt, the outcomes distribution for the potential showdown is fixed. The expected value of that showdown for the player, where the probability of scooping, quartering, tying, and getting quartered are $P_{scoop},\ P_{3Q},\ P_{tie}$, and P_{1Q}, respectively:

$$EV = pot \times P_{scoop} + pot \times 0.75 \times P_{3Q} + pot \times 0.5 \times P_{tie} + pot \times 0.25 \times P_{1Q}$$

Since the small blind player must have wagered chips to total half of the pot, the net expected value of that showdown is

$$EV = pot \times (P_{scoop} + 0.75P_{3Q} + 0.5P_{tie} + 0.25P_{1Q}) - 0.5 \times pot$$

The expression $(P_{scoop} + 0.75P_{3Q} + 0.5P_{tie} + 0.25P_{1Q})$ summarizes the player's showdown equity as a proportion of the total pot. A lookup table *summary* of these summary values can be constructed as a (16,432 × 16,432) table of values, so that $summary(A, B)$ is the normalized showdown equity of a hand of class A versus a

hand of class B. Using 64-bit floating point values, the *summary* table is just over two gigabytes in size.

Having a player's showdown equity is not enough to determine whether to play or fold; a player must also evaluate the possibility that his opponent folds, and the types of hands his opponent will play. This modelling of the opponent's strategy and the calculation of the best response against that strategy can be formalized. To do so, we define a player's *strategy vector* σ^P to be a list of probabilities, such that Player P raises or calls given that he was dealt a hand of class i with probability σ_i^P. Consider the perspective of the big blind player, dealt a hand of class B, facing a raise from the small blind. She must decide whether to call or fold, and must estimate the expected value of calling versus folding. She can do so with an estimate of the small blind's strategy vector σ^S and a calculation of *pot*, the final pot size should she call:

$$pot = 2 \times \min(\text{small blind's stack}, \text{big blind's stack}) \tag{3.1}$$

$$raise\_freq = \sum_{i=1}^{16,432} \sigma_i^S \ freq(B, i) \tag{3.2}$$

$$EV(fold) = \text{-big blind} \tag{3.3}$$

$$EV(call) = pot \times \frac{\sum_{i=1}^{16,432} \sigma_i^S \ summary(B, i) - 0.5}{raise\_freq} \tag{3.4}$$

$EV(call)$ is the expected value of playing hand B against all possible opponent hands i, weighted by the probability that the opponent actually has and plays hand i. If $EV(call)$ exceeds $EV(fold)$, the sure loss of the big blind by folding, the big blind player should call.

The small blind player faces a similar problem when deciding whether to raise or fold with a hand A. He can do so with an estimate of the big blind's strategy

vector σ^B and a calculation of *pot*:

$$EV(fold) = \text{-small blind} \tag{3.5}$$

$$EV(raise) = \sum_{i=1}^{16,432} freq(A,i)(pot(\sigma_i^B\ summary(A,i) - 0.5) + (1 - \sigma^{B,i})(\text{big blind})) \tag{3.6}$$

$EV(raise)$ is the expected value of playing hand A against all possible opponent hands i, winning the big blind if the opponent folds, and winning the expected value of the possible showdowns if the opponent calls. The `solver` program uses both *summary* and $freq$ tables as precomputed inputs for the strategy learning process.

3.5.2 Regret Matching and CFR+

`solver` implements the CFR+ variant of the CFR algorithm (Tammelin, 2014) to compute an ϵ-Nash Equilibrium for the two players using a regret minimization technique. The specific details of the CFR+ algorithm, as implemented by the `solver` program, follow.

Initialization

The `solver` program takes inputs of the $freq$ and *summary* tables, the size of the small blind and big blind, the total chip count between the two players, and an increment for the small blind's stack size. It starts by making initial estimates of the two players' strategies $\sigma^{A,t}$ and $\sigma^{B,t}$ at $t = 0$. It also initializes matrices $R_{A,i,\alpha}$ and $R_{B,i,\alpha}$, representing the cumulative regrets of A and B, dealt hand i, for playing their respective strategies instead of always playing action α. Finally, it sets the time index $t = 0$.

Calculate the expected value of Player A's current strategy

`solver` calculates the expected value $EV(\sigma^{A,t})$ of using the strategy $\sigma^{A,t}$ counterfactually on A having been dealt a hand of class i, for all 16,432 hand classes. Using the law of total probability,

$$EV(\sigma^{A,t}) = \sum_{i=1}^{16432} EV(\sigma^{A,t}|A\text{was dealt hand}i)\ \Pr(A\text{was dealt hand}i)$$

Denote $EV(\sigma^{A,t}|A\text{was dealt hand}i)$ as $EV(\sigma^{A,t}|i)$.

Calculate the expected value of Player A's alternative pure strategies

`solver` cannot calculate the expected values of all of A's 2^{16432} alternative pure strategies. Instead, as with the calculation of $EV(\sigma^{A,t})$, `solver` calculates the expected values counterfactually. Assuming that A was dealt hand i, the only strategies that can affect $EV(\sigma_0^A)$ are those where A changes his strategy given that A was dealt hand i. The two such pure strategies are Always Raise and Always Fold hand i. `solver` calculates the expected values of these two strategies using Equations 3.5 and 3.6.

The `build_sb_values` kernel implements Equations 3.5 and 3.6 using the OpenCL framework. This enables `solver` to calculate these expected values for each hand class i in parallel.

Calculate and accumulate the regrets of all pure strategies

`solver` calculates the regret of playing $\sigma^{A,t}$, conditional on being dealt hand i, compared to folding hand i, using the calculated $EV(\sigma^{A,t}|i)$ and Equation 3.5, and compared to raising using Equation 3.6. Denote these regrets compared to folding and raising as $\rho_{F,i}$ and $\rho_{R,i}$. `solver` then adds these regrets, weighted by the time index, as prescribed by CFR+, by assigning $R_{A,i,\alpha} \leftarrow R_{A,i,\alpha} + t\rho_{\alpha,i}$.

The `build_new_strategy` kernel calculates these regrets compared to Always Raise and Always Fold. It computes and accumulates them for each hand class i in parallel.

Zero out negative regrets

`solver` sweeps $R_{A,t,\alpha}$ for negative values, and if any are found, sets those values to zero. CFR+ uses this optimization to optimistically use a new best response, even if that strategy vector previously performed poorly.

Calculate Player A's regret

Having calculated the expected values for both raising and folding for all possible hand classes held by A, `solver` determines A's regret for playing $\sigma^{A,t}$ as the average regret incurred for using $\sigma^{A,t}$ for each hand class, weighted by the frequency of being dealt that class.

$$A\text{'s regret} = \sum_{i=1}^{16432} (\max(\rho_{F,i}, \rho_{R,i}) - EV(\sigma^{A,t}|i))\ \Pr(A\text{was dealt hand}i)$$

Construct a new strategy vector $\sigma^{A,t+1}$

`solver` matches A's strategy to the proportion of regrets A previously experienced, $\sigma_i^{A,t+1} = R_{A,i,raise}/(R_{A,i,raise} + R_{A,i,fold})$, as prescribed by CFR+.

Repeat the calculations above for Player B

`solver` uses a `build_bb_values` kernel that implements Equations 3.3 and 3.4 for Player B's expected values of her strategies, but otherwise follows the same algorithm.

Check if both A and B have regrets smaller than ϵ

When this is the case, the strategy vectors $\sigma^{A,t+1}$ and $\sigma^{B,t+1}$ form an 2ϵ-Nash Equilibrium, so `solver` outputs these vectors and stops. Otherwise, `solver`

increments the time index by 1 and repeats the calculations. ϵ takes the value of 0.001 chips until 10,000 iterations of the CFR+ self-play loop, after which it increases the tolerable ϵ to 0.01.

When `solver` finds an ϵ-Nash Equilibrium for the two players, it continues looping through the algorithm up to 100 further times. In many cases, `solver` finds an exact Nash Equilibrium where both players have pure strategies in this final loop. The program outputs the strategy vectors after running through this loop.

`solver` then proceeds to the next increment of the small blind's stack size. It initializes the players' strategies to be the equilibrium just discovered, on an estimate that the small adjustment to the game's circumstances will affect only a small portion of the equilibrium strategy. The `solver` continues constructing equilibrium strategies for the higher small blind stack sizes until the small blind stack size is more than half the total chips in the game. When the small blind has more than half the total chips, the big blind has correspondingly less than half. Based on Equation 3.1, the potential pot size becomes capped by the big blind's stack, to pot sizes that `solver` previously analyzed. The previously calculated equilibrium for the small blind's bigger stack size is the same as for the small blind's smaller stack size. `solver` calculated ϵ-Nash Equilibrium strategies for games with a total of 9,000 chips, with table stakes of 1000/2000, 750/1500, 600/1200, 500/1000, 400/800, 300/600, 250/500, 200/400, 150/300, 100/200, and 50/100, for stack sizes of the small blind from 50 to 8,950 chips in increments of 50 chips.

The `verify` program calculates the value of the game given the strategy vectors of the small and big blinds, and optionally also flag strategy choices that are not the best response to the opponent's strategy. The `build_best_response` program constructs the best response strategy vector against an opponent's given strategy vector.

3.6. Feature Valuation

The output of the Equilibrium Solver is an ϵ-Nash Equilibrium strategy profile for the two players, prescribing how frequently a player should raise or call for every possible Omaha hand type. But with 16,432 different Omaha hand types, it would be difficult to memorize even one strategy table for a particular stack size situation. The Feature Valuation process assigns scores to different attributes of an Omaha hand type so that a player can calculate a Power Level score for that hand.

The `HandAttributes.py` program cycles through the different possible hand classes and identifies attributes about the four cards in a hand of each class. This program tags hands across the hand attributes in Table 3.4, setting the attribute value to 1 if the hand contains the attribute. The program can also exclude from its output the hands that contain particular attributes. It outputs two comma-separated values files. The first file, `attribs_separate.csv`, lists hand classes and their attributes tags. The second file, `attribs_combined.csv`, extends the output of the first file with a dummy 0/1 variable column to identify whether the hand is to be analyzed in the position of the small or big blind.

Separately, `cutoff.py` calculates the playability of each hand class reported in an output file from `HandAttributes.py`. This program uses a simple measure of a hand's playability: the maximum stack size to big blind ratio where a player should play that hand. This ratio R is the same ratio used by the SAGE System for Texas Hold'em Poker. It is an appropriate measure through its relationship with the pot odds faced by each player: as the players' stack size increases, the number of chips at risk when raising or calling increases, and the pot odds offered approaches 1:1. Hands that win infrequently require higher pot odds for them to be profitable to play; this corresponds to smaller stack sizes and lower R values. This measure

Attribute	Code	Example Hand	Attribute & Value
Single card of face value X, for Nines to Kings	has$X$	A♠ K♠ K♡ J♣	hasK = 0, hasJ = 1
Single card of Lo card X, hand cannot make Lo	bare$X$	K♠ K♡ J♣ 2♠	bare2 = 1
Single card of Lo card X, hand can make Lo	hasLow$X$	A♠ A♠ K♡ 3♣	bareA = 0, hasLowA = 0, hasLow3 = 1
Pair of face value X	hasPair$X$	A♠ K♠ K♡ 3♣	hasPairK = 1
Exactly one pair	is_one_pair	A♠ K♠ K♡ 3♣	is_one_pair = 1
Exactly two pairs	is_two_pair	A♠ A♣ K♠ K♡	is_one_pair = 0, is_two_pair = 1
Three of a kind	is_trips	A♠ K♠ K♡ K♣	is_one_pair = 0, is_trips = 1
Four of a kind	is_quads	K♠ K♡ K♢ K♣	is_one_pair = 0, is_two_pair = 0, is_trips = 0, is_quads = 1
Double-suited	suited_AABB	A♠ K♠ Q♡ J♡	suited_AABB = 1
Mono-suited	suited_AAAA	A♠ K♠ Q♠ J♠	suited_AAAA = 1
Single-suited with one off-suit card	suited_AAAB	A♠ K♠ Q♠ J♡	suited_AAAB = 1
Single-suited with two off-suit cards	suited_AABC	A♠ K♠ Q♡ J♣	suited_AABC = 1
Lo cards with exactly X distinct Lo face values	low_card_X	A♠ 4♠ 3♡ 3♣	low_card_3 = 1
Suited with X-high in suit	suited$X$	A♠ K♠ Q♡ J♣	suitedA = 1, suitedK = 0
Suited connector XY	cXYs	A♠ K♠ Q♡ J♡	cQJs = 1
Connector XY	cXY	K♠ Q♡ J♡ T♢	cQJ = 1, cJT = 1
Range of X between highest and lowest card	range$X$	K♠ Q♡ J♡ 9♢	range5 = 1

Table 3.4: Attributes of Omaha Hi-Lo four-card hands

assumes that hands that should be played at high R values should also be played at low R values. In practice, this assumption holds for nearly all hand classes at R levels up to 10. `cutoff.py` reports the maximum R levels for the hand classes in three output files, multiplied by ten for convenience: small blind R levels, big blind R levels, and combined. The first two outputs align with the `attribs_separate.csv` file from `HandAttributes.py`, and the last file aligns with `attribs_combined.csv`.

The Feature Valuation process uses the list of independent attribute variables in the `attribs_combined.csv` and the corresponding R level vector in the `attribs_combined.csv` file. It uses linear regression to estimate the effect each attribute has on the overall playability of all the analyzed hands. The linear regression coefficients form a starting estimate for the Power Index calculation rules.

Unlike the previous components of this project, determining the ideal Feature Valuation coefficients is qualitative in nature. Using fewer rules in the calculation of the Power Index heuristic increases its usability but decreases the accuracy relative to the exact Nash Equilibrium strategy. The results of Feature Valuation are values of different hand attributes. A player can calculate the sum of his hand's attributes' values, and on comparing that sum with a cutoff score, decide whether to raise or fold his hand.

Chapter 4: Results

This chapter will discuss the results of the Equilibrium Solver and Feature Valuation described in the Methodology chapter. First, it describes characteristics of the ϵ-Nash Equilibrium and compares them to other research results. It then discusses the results of the linear regression and the analysis that followed the regression calculation, culminating in the Omaha Raise All-in/Call Lightweight Endgame (ORACLE) Strategy scoring heuristic. Finally, this chapter discusses the ORACLE Strategy in detail, comparing it to the ϵ-Nash Equilibrium and analyzing its deviation from optimal play.

4.1. Analysis of the ϵ-Nash Equilibrium

Lee Jones, the creator of the SAGE System, claimed that "most players play far too tightly in heads-up jam-or-fold situations" (Jones, 2006). While analysing this claim is outside the scope of this thesis, it is possible to quantify how frequently a player should play in these situations. A high play frequency in a strategic equilibrium supports Jones' claim that players likely do not play frequently enough.

Both the small blind and big blind should play many hands at the low R levels that signify that a player is running out of chips. In many endgame situations, the R level is ten or below, leaving both players at the mercy of the cards as they play more than 70% of hands each. The optimal play frequency drops off as R increases,

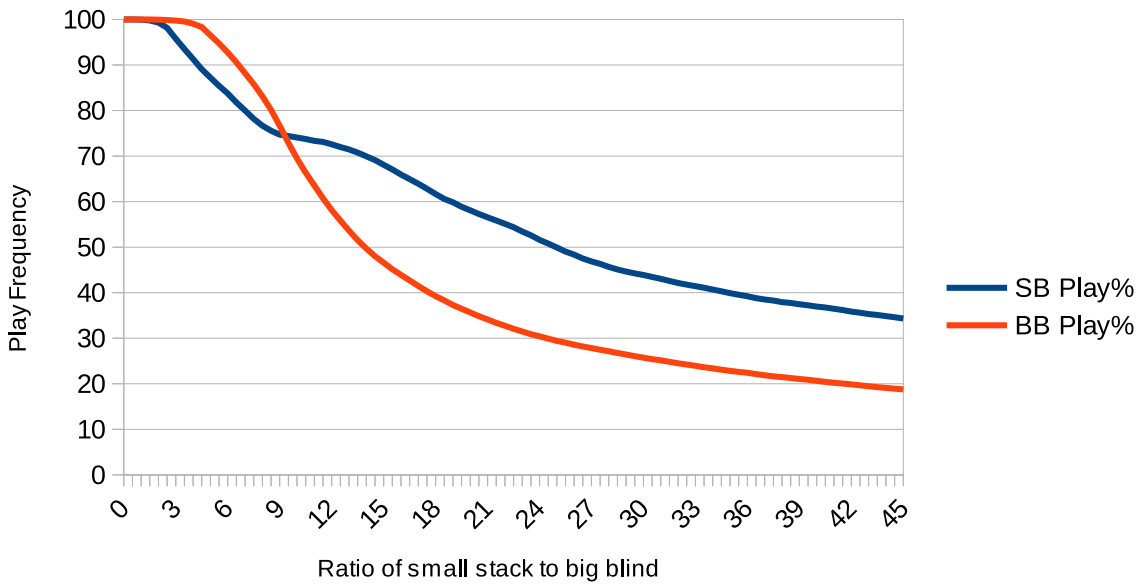

Figure 4.1: Optimal Play Frequency by the small and big blinds, as a function of R

as expected, as raising all-in requires putting more chips at risk. The small blind's play frequency falls to just over 33% at an $R = 45$, and the big blind plays just over half that, at 18%. This can be understood as a consequence of pot odds: as R increases, the pot odds offered to the big blind approaches 1:1, and the big blind cannot profitably call with hands that win less than half the time.

The equilibrium results clearly demonstrate the value of Aces in Omaha Hi-Lo Poker. The worst possible hand that contains two Aces is the four-of-a-kind Ace hand. A player with this hand will always use two Aces from the hand, so he cannot make any straights, any flush, and cannot win any Lo pot. It is playable, based on the playability measure described in Section 3.6, even at the lowest table stakes analyzed, 50/100, $R = 45$. For comparison, the four-of-a-kind Kings hand has a playability measure of less than two, and the least playable hand with a pair of Kings, K♠ K♡ 9♢ 2♣, should not be played by the small blind at $R \geq 16$. Aces are also valuable as the best Lo card possible; aside from the hands with three-of-

Attribute	Least playable hand for small blind	R level, small blind	R level, big blind
Two or more Aces	A♠ A♡ A♢ A♣	45+	8.5
Two Kings	K♠ K♡ 9♢ 2♣	15	10
Two Queens	Q♠ Q♡ 9♢ 2♣	6	7.5
Ace and Lo card, not trips	A♠ T♡ 9♢ 8♣	18.5	14
Ace and Pair	A♠ T♡ 9♢ 9♣	5	7.5
Double-suited	K♠ 2♠ 9♡ 2♡	2	4
Double-suited with Ace	A♠ 9♠ T♡ 9♡	38.5	17
Two Pairs	9♠ 9♡ 2♢ 2♣	2	4
Double-suited Two Pairs	9♠ 9♡ 2♠ 2♡	4.5	7
Double-suited and three different Lo cards	Q♠ 8♠ 7♡ 3♡	27	14.5

Table 4.1: Playability of selected hands by attribute

a-kind, the least playable hand with an Ace and another Lo card, A♠ T♡ 9♢ 8♣, remains playable for the small blind at $R \leq 18$. Table 4.1 lists other hand attribute combinations and their playability measures.

Hands that are readily playable by the small blind are not necessarily strong for the big blind. The small blind can play all hands that are an "Ace and Lo card, not three of a kind" up to $R \leq 18$. The least playable hand of this type for the small blind is A♠ T♡ 9♢ 8♣. But the least playable hand of this type for the big blind is A♠ 9♡ 2♢ 2♣, which she should fold at $R > 12.5$. There is therefore no method to rank either hand as superior to the other. Conversely, there are hands that are more playable as the big blind than as the small blind. The hand J♠ 6♠ T♡ 9♡ is only profitable to play as the small blind at $R \leq 3.5$ but remains playable for the big blind when $R \leq 6.5$.

Similar to the two player Texas Hold'em jam/fold equilibrium, not all hands that are playable with a large stack are playable with a small stack (Miltersen & Sørensen, 2007). At table stakes of 200/400, 8♠ 5♠ 6♡ 2♣ should be raised by the

small blind if either the small blind or big blind has 3,500 or fewer chips, or if both small and big blinds have more than 4,100 chips. Similarly, A♠ K♠ Q♠ J♠ should be raised by the small blind if either player has 2,650 or fewer chips, or both players have more than 3,950 chips. These discontinuities show that there is no uniform playability ranking across the hand types, illustrated in Figure 4.2. There are 116 such discontinuities at table stakes of 200/400, but only three of these persist to stakes of 250/500, and none to 300/600. As described in Section 3.6, the Feature Valuation assumes that no discontinuities exist; while this assumption is not valid at 200/400, its impact, affecting fewer than 1% of hand classes, is small.

Hand	3,000	4,000
8♠ 5♠ 6♡ 2♣	Raise	Fold
A♠ K♠ Q♠ J♠	Fold	Raise

Figure 4.2: Non-dominance of hands at stacks of 3,000 and 4,000, table stakes 200/400

The unexploitability of a strategy does not guarantee its profitability. At very low R levels, both players play nearly all their hands, effectively determining the winner of any given hand by random luck. This results in a near-zero expected value, when both players play whatever cards they were dealt. Figure 4.3 illustrates the minimax of the small blind's jam/fold strategy, showing that it peaks at $R = 9.5$, and falls below zero by $R = 16$. The SAGE System is also only profitable for the small blind at $R \leq 7$ (Jones, 2006).

4.2. Value of Hand Features

Based on the playability cutoffs listed in Table 4.1, four types of hands were removed from the analysis, as their exact cutoff levels are too high. To model an endgame situation, the playability measures were calculated using table stakes of

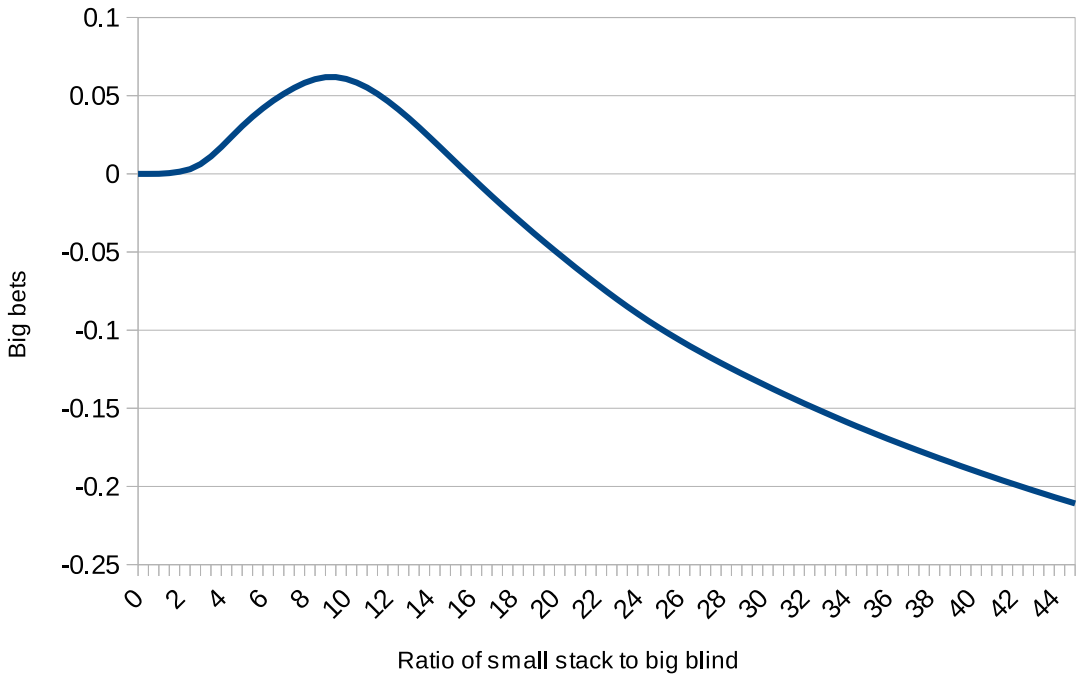

Figure 4.3: Minimax value for the small blind, as a function of R

150/300. As a result, any hand that is playable above $R = 15$ will have a reported playability metric of 150. This artificial cutoff causes the value of strong hands such as a Pair of Aces to have an understated playability value, with their corresponding attributes also undervalued. The regression results are also not expected to be effective at the cutoff value $R = 15$. Instead, the model is tested at table stakes of 200/400, corresponding to $R = 11.25$. Figure 4.4 shows the results of the linear regression across the attributes described in Table 3.4. The four hand types excluded from the regression analysis are:

- Hands with two or more Aces
- Hands with a Pair of Kings
- Hands with an Ace and a Lo card

- Double-suited hands with an Ace

The use of a zero intercept accounts for the linearly dependent suitedness attributes: a hand will always be one of double-suited, single-suited with two off-suit cards, single-suited with one off-suit cards, mono-suited, or unsuited. The linear regression results show that the dominant attributes of a hand's playability are the presence of an Ace, the presence and value of any pairs, the suitedness of the hand, and the presence and quantity of low cards. The connectors and suited connectors only have a small effect on the overall playability of a hand, as do the specific high card values of suited hands. Removing these attributes from the regression leaves a smaller linear model, as shown in Figure 4.6.

The correlation coefficient drops from 0.9645 to 0.9637, still a very explanatory model. From here, the valuation rules for each attribute can be read off the regression coefficients. The resultant model tracks the equilibrium strategy very closely, as seen in Figure 4.5. Even while the linear model strategy is not identical to the equilibrium, it is nevertheless difficult to exploit, as shown by the small gap between the ideal equilibrium curve and the two Exploitative strategy curves in Figure 4.5. The small blind using this model potentially gives up 6.5 chips, approximately 0.016 big blinds, to a maximally exploitative opponent, and the big blind possibly gives up 0.01 big blinds.

But while the linear model strategy is accurate, it is not easy to use. It is not reasonable to memorize these coefficients and calculate the playability indices at a live poker table. The ORACLE Strategy uses further simplifications by rounding coefficients and grouping attributes of similar type and value into one aggregate score.

```
              Estimate Std. Error t value Pr(>|t|)
suited_AABB   75.06386    4.58868  16.358  < 2e-16 ***
suited_AABC   51.06602    4.57024  11.174  < 2e-16 ***
suited_AAAB   45.61450    4.58538   9.948  < 2e-16 ***
suited_AAAA   39.00056    4.64692   8.393  < 2e-16 ***
suited_ABCD   16.23615    4.53929   3.577 0.000349 ***
hasBareA      45.56065    1.74301  26.139  < 2e-16 ***
hasBare2      -5.56085    1.71359  -3.245 0.001176 **
hasBare3      -4.80332    1.71308  -2.804 0.005053 **
hasBare4      -4.32431    1.70956  -2.529 0.011429 *
hasBare5      -2.66168    1.70024  -1.565 0.117486
hasBare6       0.01130    1.68819   0.007 0.994661
hasBare7       2.44854    1.68776   1.451 0.146861
hasBare8      10.59496    1.72326   6.148 7.96e-10 ***
has9          -9.26773    1.25062  -7.411 1.30e-13 ***
hasT          -1.43792    1.25580  -1.145 0.252210
hasJ          -0.46050    1.29097  -0.357 0.721313
hasQ           6.55489    1.29541   5.060 4.22e-07 ***
hasK          13.03615    1.28035  10.182  < 2e-16 ***
hasLowA             NA         NA      NA       NA
hasLow2       15.65821    1.67597   9.343  < 2e-16 ***
hasLow3       16.17328    1.66254   9.728  < 2e-16 ***
hasLow4       17.13257    1.65114  10.376  < 2e-16 ***
hasLow5       19.15817    1.65312  11.589  < 2e-16 ***
hasLow6       14.07537    1.66329   8.462  < 2e-16 ***
hasLow7       -0.86279    1.66877  -0.517 0.605149
hasLow8       -8.00539    1.68573  -4.749 2.06e-06 ***
hasPairA            NA         NA      NA       NA
hasPair2     -15.58783    2.59239  -6.013 1.85e-09 ***
hasPair3      -6.70535    2.58999  -2.589 0.009633 **
hasPair4       2.93011    2.58869   1.132 0.257692
hasPair5      14.41827    2.59043   5.566 2.64e-08 ***
hasPair6      20.56876    2.59449   7.928 2.33e-15 ***
hasPair7      20.34035    2.59690   7.833 4.98e-15 ***
hasPair8      25.69891    2.60009   9.884  < 2e-16 ***
hasPair9      22.22332    2.57578   8.628  < 2e-16 ***
hasPairT      42.00693    2.57345  16.323  < 2e-16 ***
hasPairJ      56.35528    2.57899  21.852  < 2e-16 ***
hasPairQ      73.08115    2.57213  28.413  < 2e-16 ***
hasPairK            NA         NA      NA       NA
is_one_pair   -5.88005    0.71409  -8.234  < 2e-16 ***
is_two_pair         NA         NA      NA       NA
is_trips     -33.24601    3.59013  -9.260  < 2e-16 ***
is_quads            NA         NA      NA       NA
low_card_2    46.53096    1.88152  24.731  < 2e-16 ***
low_card_3    45.07704    3.25715  13.839  < 2e-16 ***
low_card_4     3.75768    4.80952   0.781 0.434633
suitedA             NA         NA      NA       NA
suitedK        0.71356    0.64838   1.101 0.271114
suitedQ       -0.17266    0.61979  -0.279 0.780567
suitedJ        0.53166    0.60954   0.872 0.383092
c65           -7.26607    0.87080  -8.344  < 2e-16 ***
c76           -0.63301    0.87765  -0.721 0.470759
c87            9.32823    0.87721  10.634  < 2e-16 ***
c98           -9.83669    0.85999 -11.438  < 2e-16 ***
cT9           -3.48146    0.84058  -4.142 3.46e-05 ***
cJT            3.45460    0.85248   4.052 5.09e-05 ***
cQJ            1.60290    0.86622   1.850 0.064262 .
cKQ           -5.72949    0.87836  -6.523 7.03e-11 ***
c65s           3.51241    1.14183   3.076 0.002100 **
c76s           2.33150    1.14138   2.043 0.041094 *
c87s          -0.37413    1.14141  -0.328 0.743084
c98s           0.03762    1.14139   0.033 0.973706
cT9s          -1.02645    1.11408  -0.921 0.356879
cJTs          -0.13021    1.15111  -0.113 0.909942
cQJs          -0.89316    1.14671  -0.779 0.436051
cKQs          -1.74516    1.19169  -1.464 0.143086
range2        11.98542    2.13874   5.604 2.12e-08 ***
range3        22.37658    1.23473  18.123  < 2e-16 ***
range4        16.87906    0.91810  18.385  < 2e-16 ***
range5        15.24941    0.75451  20.211  < 2e-16 ***
range6         5.05374    0.64542   7.830 5.08e-15 ***
range7         5.95275    0.54663  10.890  < 2e-16 ***
is_big_blind  -4.55564    0.29052 -15.681  < 2e-16 ***
---
Residual standard error: 22.23 on 23345 degrees of freedom
Multiple R-squared:  0.9646,    Adjusted R-squared:  0.9645
F-statistic:  9494 on 67 and 23345 DF,  p-value: < 2.2e-16
```

Figure 4.4: Linear Regression results on playability metric, zero intercept, all attributes

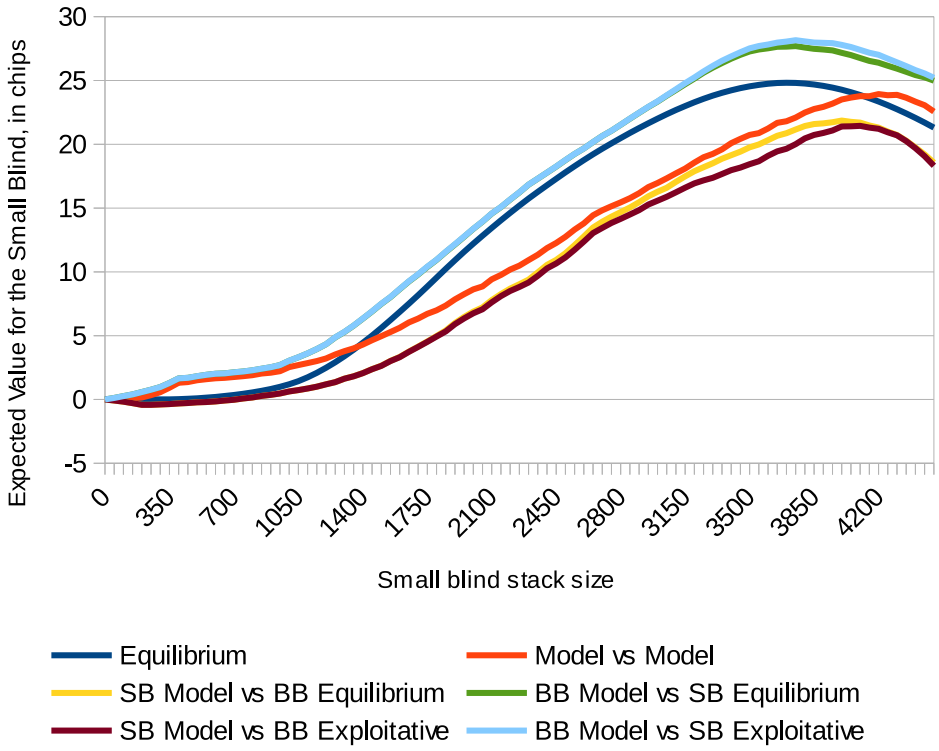

Figure 4.5: Strategy performance between equilibrium, model, and maximal exploitation of the model strategy at table stakes 200/400

4.3. The ORACLE Strategy

The Omaha Raise All-in/Call Lightweight Endgame Strategy is a newly-developed scoring system for a player's four-card Omaha Hi-Lo hand. A player can calculate a Power Index using this system to determine whether to raise/call or fold, as follows:

- If the hand satisfies any of the following, Raise as the Small Blind and Call as the Big Blind.
  - The hand contains two or more Aces
  - The hand contains exactly two Kings
  - The hand contains an Ace and any Lo card (any card from Two to Eight)

```
             Estimate Std. Error t value Pr(>|t|)
suited_AABB   75.4023     4.6251  16.303  < 2e-16 ***
suited_AABC   51.2397     4.6169  11.098  < 2e-16 ***
suited_AAAB   45.9329     4.6233   9.935  < 2e-16 ***
suited_AAAA   39.4743     4.6691   8.454  < 2e-16 ***
suited_ABCD   16.2362     4.5896   3.538 0.000405 ***
hasBareA      45.9749     1.7567  26.171  < 2e-16 ***
hasBare2      -6.1151     1.7278  -3.539 0.000402 ***
hasBare3      -5.3562     1.7273  -3.101 0.001932 **
hasBare4      -4.8630     1.7240  -2.821 0.004795 **
hasBare5      -3.1391     1.7153  -1.830 0.067257 .
hasBare6      -0.2991     1.7039  -0.176 0.860649
hasBare7       2.4725     1.7025   1.452 0.146429
hasBare8       5.9118     1.6902   3.498 0.000470 ***
has9         -12.2450     1.2329  -9.932  < 2e-16 ***
hasT          -0.8358     1.2271  -0.681 0.495808
hasJ           1.8411     1.2222   1.506 0.131979
hasQ           5.7257     1.2187   4.698 2.64e-06 ***
hasK          12.2488     1.2153  10.079  < 2e-16 ***
hasLowA           NA         NA      NA       NA
hasLow2       14.2839     1.6790   8.508  < 2e-16 ***
hasLow3       14.8790     1.6648   8.937  < 2e-16 ***
hasLow4       15.9166     1.6525   9.632  < 2e-16 ***
hasLow5       16.1336     1.6428   9.821  < 2e-16 ***
hasLow6       11.1568     1.6369   6.816 9.60e-12 ***
hasLow7        0.9696     1.6366   0.592 0.553548
hasLow8       -8.7167     1.6430  -5.305 1.13e-07 ***
hasPairA          NA         NA      NA       NA
hasPair2     -16.0038     2.6146  -6.121 9.46e-10 ***
hasPair3      -7.0042     2.6112  -2.682 0.007315 **
hasPair4       2.7465     2.6086   1.053 0.292426
hasPair5      13.1576     2.6074   5.046 4.54e-07 ***
hasPair6      19.4988     2.6078   7.477 7.86e-14 ***
hasPair7      22.4482     2.6099   8.601  < 2e-16 ***
hasPair8      25.7124     2.6099   9.852  < 2e-16 ***
hasPair9      20.8810     2.5930   8.053 8.47e-16 ***
hasPairT      43.2602     2.5860  16.729  < 2e-16 ***
hasPairJ      58.9256     2.5782  22.856  < 2e-16 ***
hasPairQ      73.0455     2.5694  28.429  < 2e-16 ***
hasPairK          NA         NA      NA       NA
is_one_pair   -5.8403     0.7210  -8.100 5.75e-16 ***
is_two_pair       NA         NA      NA       NA
is_trips     -32.1351     3.6230  -8.870  < 2e-16 ***
is_quads          NA         NA      NA       NA
low_card_2    48.1883     1.8792  25.643  < 2e-16 ***
low_card_3    48.8163     3.2399  15.067  < 2e-16 ***
low_card_4    10.1430     4.7635   2.129 0.033237 *
range2         9.9900     2.1258   4.699 2.62e-06 ***
range3        20.5774     1.1964  17.199  < 2e-16 ***
range4        15.4809     0.8847  17.498  < 2e-16 ***
range5        14.1812     0.7355  19.282  < 2e-16 ***
range6         4.3286     0.6350   6.817 9.52e-12 ***
range7         5.7728     0.5397  10.695  < 2e-16 ***
is_big_blind  -4.5556     0.2937 -15.509  < 2e-16 ***
---
Residual standard error: 22.47 on 23364 degrees of freedom
Multiple R-squared:  0.9638,    Adjusted R-squared:  0.9637
F-statistic: 1.295e+04 on 48 and 23364 DF,  p-value: < 2.2e-16
```

Figure 4.6: Linear Regression results on playability metric, filtered attributes

  - The hand contains an Ace and is double-suited

- Score the suit distribution of the hand.
  - If the hand is unsuited, start with a Power Level of 16
  - If the hand is single-suited, start with a Power Level of 40
  - If the hand is double-suited, start with a Power Level of 75
  - If the hand is single-suited, add 6 for every card in the hand not in that suit

- Score the unpaired cards in the hand.
  - For an unpaired Ace, King, or Queen, add 46, 12, or 6, each
  - For an unpaired 9, subtract 12
  - For an unpaired 2, 3, or 4, if there are no other Lo cards in the hand, subtract 5

- Score the low cards in the hand.
  - If the hand has fewer than two Lo cards (Two to Eight) of different value, skip this section.
  - If the hand contains four Lo cards all of different face values, add 10. Otherwise, add 48.
  - Add 15 for each different face value of Lo card from Two to Five, and 10 for a Six
  - Subtract 8 if the hand contains an Eight

- Score the pairs in the hand. Three of a kinds do not count as pairs in this category.

  - For each pair of Twos or Threes, subtract 16 and 8, respectively
  - For each pair of Sixes to Nines, add 20. For a pair of Fives, add 15
  - For each pair of Tens, Jacks, or Queens, add 45, 60, and 75, respectively

- Score the hand's structure.
  - If the hand contains exactly one pair, subtract 6
  - If the hand contains a three of a kind (but not a four of a kind), subtract 32

- Score the hand's straight potential.
  - Calculate the distance from the highest to the lowest card in the hand.
  - If the distance is two (the top and bottom cards are consecutive face values), add 10
  - If the distance is three, add 20
  - If the distance is four or five, add 15
  - If the distance is six or seven, add 5

- If you are the Big Blind, subtract 5
- Calculate the action threshold $\texttt{10R} = \frac{\text{size of the small stack} \times 10}{\text{size of the big blind}}$.

If the Power Level exceeds `10R`, Raise as the Small Blind and Call as the Big Blind.

4.4. Analysis of the Strategy

The ORACLE Strategy is calibrated to play at a maximum R level of 11.25, corresponding to table stakes of 200/400 and 9,000 total chips in the tournament.

Changing the table stakes used for the cutoff will change the regression coefficients, calibrating the strategy to higher or lower R level ranges. As a further approximation of the linear regression approximation, the ORACLE strategy is less accurate still, with a correlation coefficient of 0.9577. But while goodness of fit is useful, the real test of this model is the corresponding strategy and its exploitability.

```
                Estimate Std. Error t value Pr(>|t|)
oracle_strategy  0.987292   0.001802  547.94   <2e-16
is_big_blind    -4.869797   0.303881  -16.02   <2e-16
---
Residual standard error: 24.25 on 23410 degrees of freedom
Multiple R-squared:  0.9577,    Adjusted R-squared:  0.9577
F-statistic: 2.652e+05 on 2 and 23410 DF,  p-value: < 2.2e-16
```

Figure 4.7: Linear regression on ORACLE Strategy versus playability metric

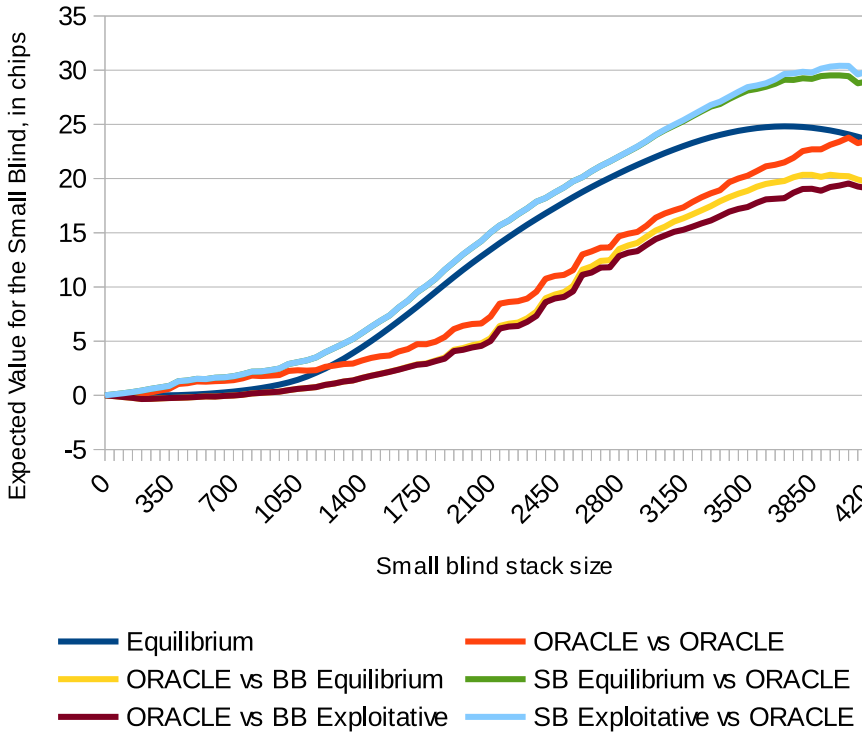

Figure 4.8: The ORACLE Strategy performance at 200/400

The ORACLE strategy, with more strategic deviations from the equilibrium than the attribute model coefficients, is more exploitable. But it is still a very strong strategy. The small blind using the ORACLE System is worse by 8.7 chips against an equilibrium big blind opponent, at a stack size of 2,300 chips, compared to using the equilibrium strategy himself. In turn, the big blind using the ORACLE System is worse by 6 chips against an equilibrium small blind, at a stack size of 4,500 chips. Against maximally exploitative opponents, using the ORACLE System costs the small blind at most 8.96 chips, and the big blind at most 6.87 chips compared to perfect play. These correspond to 0.0224 and 0.0172 big blinds.

Chapter 5: Summary and Conclusions

This thesis project sought to compute an approximate Nash Equilibrium strategy for No-Limit Jam/Fold Omaha Hi-Lo Hold'em Poker, and construct an evaluation heuristic for an Omaha Hi-Lo poker hand that approximates the Nash Equilibrium strategy. Through this project's Equilibrium Solver and Feature Valuation components, an ϵ-Nash Equilibrium was found and the ORACLE Strategy heuristic was constructed. The objectives of this thesis were thus successfully met.

5.1. Contributions

In addition to computing the ϵ-Nash Equilibrium strategy for jam/fold Omaha Hi-Lo Poker, this project contributes to artificial intelligence research by demonstrating the use of the CFR+ algorithm and the use of OpenCL and GPU acceleration for analyzing games. The CFR+ algorithm has proven itself in Limit Texas Hold'em, a game with at most 169 possible hand types per player. This project found that CFR+ is also effective in games with many more possible player hands, and achieves similar performance gains over plain CFR for these games as well. This project also shows that GPU parallel processing using OpenCL can achieve performance gains even in a turn-based game such as poker. While using OpenCL did not reach the hundred-fold speed improvements observed by previous research (Stone et al., 2010), a 20% improvement was achieved using GPU processing alone.

5.2. Lessons Learned

As the previous chapters have shown, this project resulted in a nearly unexploitable strategy and scoring heuristic for Omaha Hi-Lo poker, fulfilling all its proposed objectives. There were many lessons learned, but two lessons feature most prominently.

The first lesson is the importance of preparation and the value of theory. This featured most prominently in the `MatchupTabulator` task of evaluating all $\binom{52}{8}$ eight-card combinations. A brute force attempt would require almost one year of computation. Enumerating the symmetric classes reduced this to twelve days. Careful preparation and memory caching reduced the number of $HandValue$ memory lookups by one thousand-fold. Theory also reduced the Equilibrium `solver` time: using the new CFR+ enhancements resulted in a convergence to an equilibrium strategy in, on average, half the number of self-play iterations compared to vanilla CFR.

The second lesson is the importance of resource management. Originally, the data consolidation step described in Section 3.4 was planned to run in the Relational Database Service (RDS) by Amazon. But this became prohibitive in both cost and system load. In particular, the database size requirement expanded far beyond expectations and disk access was slower than anticipated. The slow disk write speed slowed down the cluster of `GameEvaluator` nodes, as the web server itself faced delays in writing results to the database and retrieving new work packets. Switching to a local SQLite database and storing results in an uncollated format on Simple Storage Service dramatically improved response times. Using S3 instead of RDS also simplified disk space allocation, as S3 scales automatically with usage.

5.3. Known Issues and Future Work

There are a few issues in this analysis of Omaha Hi-Lo Poker. The major issue is the assumption of jam-or-fold play, which is only realistic in tournament endgame situations. In a cash game or the early stages of a poker tournament, using the jam/fold equilibrium is unprofitable as the small blind, as the R ratio typically exceeds 50.

This analysis also assumes risk-neutral players that follow the Independent Chip Model. Miltersen and Sørensen explicitly assumed otherwise, modeling all possible stake levels and plays together with a goal of winning the tournament. They could construct their model for Texas Hold'em poker because that game has only win, lose, or tie results in each round, and therefore could guarantee that their players' stack sizes remained multiples of 50 chips at all times. In Omaha Hi-Lo, a player can win one-quarter or three-quarters of the pot, violating this game assumption. Based on the assumptions made, the computed equilibrium strategy achieves maximum chips won in each round. It is therefore an accurate short-stacked cash game strategy. Further work is necessary to validate or repudiate the ICM and risk-neutral player assumptions for the iterated game structure of a tournament.

Finally, the ORACLE Strategy, while short, still consists of seven hand category rules. The SAGE System for Texas Hold'em uses four rules. Further refinements on hand attributes or scores might achieve similarly strong results with fewer rules; these other attributes have not been explored.

References

Bowling, M., Burch, N., Johanson, M., & Tammelin, O. (2015). Heads-up limit holdem poker is solved. *Science*, 347(6218), 145–149.

Ganzfried, S. & Sandholm, T. (2008). Computing an approximate jam/fold equilibrium for 3-player no-limit Texas Hold'em tournaments. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2* (pp. 919–925).: International Foundation for Autonomous Agents and Multiagent Systems.

Harrington, D. & Robertie, B. (2010). *Harrington on Online Cash Games: 6-Max No-Limit Hold'em*. Two Plus Two Publishing LLC.

Hart, S. & Mas-Colell, A. (2000). A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5), 11271150.

Hutchison, E. (1997). Hutchison point count system for omaha high-low poker.

Jones, L. (2006). Are you sage? getting an edge in heads-up no-limit Hold'em. *Card Player Magazine*, 19(2).

Miltersen, P. B. & Sørensen, T. B. (2007). A near-optimal strategy for a heads-up no-limit Texas Hold'em Poker tournament. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems* (pp. 191).: ACM.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1), 48–49.

Robinson, J. (1951). An iterative method of solving a game. *Annals of mathematics*, (pp. 296–301).

Sklansky, D. (2007). *Tournament Poker for Advanced Players*. Two Plus Two Publishing LLC.

Sklansky, D. & Malmuth, M. (1999). *Hold'em Poker For Advanced Players*. Two Plus Two Publishing LLC.

Stone, J. E., Gohara, D., & Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(1-3), 66–73.

Tammelin, O. (2014). Solving large imperfect information games using CFR+. *arXiv preprint arXiv:1407.5042*.

Von Neumann, J. & Morgenstern, O. (2007). *Theory of games and economic behavior*. Princeton university press.

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2007). Regret minimization in games with incomplete information. *Advances in neural information processing systems*, (pp. 1729–1736).

Appendix A: Glossary

ϵ-Nash Equilibrium A strategy profile where no player can improve their expected score by changing their own strategy by more than a small value ϵ, assuming no other players change their strategy. vi, 2, 19, 21, 22, 46, 49, 62

AWS Amazon Web Services. 4, 22–24, 38

best response the strategy available to a player that maximizes his expected return, assuming that all other players' strategies remain fixed. 18, 41, 44, 45

big blind The larger of the forced bets at the start of the game, made by the player left of the small blind. Also, the player that must make the big blind bet. vii, 5, 6, 9–12, 16, 41, 42, 45, 46, 48–52, 54, 59, 61

CFR Counterfactual Regret Minimization. 4, 22, 42, 62, 63

connector Two cards with consecutive face value, valuable for constructing straight hands. 14, 47

CPU central processing unit. 23, 35

double-suited An Omaha poker hand with its cards in one of two suits, two cards to each suit. Example: A♣ 5♣ Q♠ 8♠. 14, 47, 51, 54, 58

EC2 Elastic Compute Cloud. 23, 24, 68, 69

flush Five cards, all with the same suit. Example: K♣ 8♣ 6♣ 3♣ 2♣. 7, 8, 13–15, 29, 50

four-flush Four cards, all with the same suit. 37

four-of-a-kind Four cards sharing the same face value. 8, 13, 50

full house A poker hand consisting of both a three-of-a-kind and a pair. Example: J♣ J♡ J♠ Q♢ Q♠. 8

GPGPU general purpose computing on graphics processing units. 24

GPU graphics processing unit. 23, 24, 35, 38, 39, 62

Hi The half of the pot set aside, whose winner is the player with the highest-ranking poker hand. 7, 8, 14, 27, 28, 31, 32, 34, 35, 69

ICM Independent Chip Model. 39, 40, 64

Lo The half of the pot set aside, whose winner is the player with the lowest-valued five cards of distinct face value, all Eight or below. 4, 7, 13–15, 27, 28, 31, 32, 34, 35, 47, 50, 51, 53, 56, 58, 69

Nash Equilibrium A strategy profile where no player can improve their expected score by changing their own strategy, assuming no other players change their strategy. 2, 4, 19–21, 62, 69

on-demand instance An EC2 instance run on demand by a user, with fees charged according to the published fixed price structure. 23, 38

OpenCL Open Compute Language, a framework to facilitate computation across different hardware types, such as CPUs, GPUs, and other heterogeneous hardware. 4, 22–24, 35, 43, 62

quarter One player solely winning one of the Hi or Lo pots, and splitting the other, leaving the other player with a quarter of the overall pot. 9, 35, 38, 40

RDS Relational Database Service. 63

S3 Simple Storage Service. 23, 36, 38, 63

SAGE System Sit-And-Go Endgame System, an approximate Nash Equilibrium strategy for Texas Hold'em tournament endgames. 21, 22, 46, 49, 52, 64

scoop One player winning both Hi and Lo pots. 7, 38, 40

single-suited An Omaha poker hand with at least two cards having the same suit. Example: K♣ Q♣ T♣ 4♡. 14, 47, 54, 58

small blind The smaller of the forced bets at the start of the game, made by the player left of the button. Also, the player that must make the small blind bet. vii, 5, 9–12, 17, 35, 40–42, 45, 46, 48–54, 59, 61

spot instance An EC2 instance dynamically available according to the user's maximum bid price, compared to the spot market rate. If the bid exceeds the market rate, the instance starts, and fees accrue according to the spot rate. If the market rate later exceeds the bid, the instance stops.. 23, 38

straight Five cards, with consecutive face value. Example: K♣ Q♡ J♡ T♢ 9♣. 8, 13, 50, 67

straight flush Five cards, with consecutive face value, all with the same suit. Example: K♣ Q♣ J♣ T♣ 9♣. 8, 14, 28, 29

suited Two cards having the same suit. 47

three-flush Three cards, all with the same suit. 37

three-of-a-kind Three cards sharing the same face value. 8, 13, 37

two-flush Two cards having the same suit. 37

unsuited Cards all with different suits. 37, 54, 58