Final Project Introduction - Texas Hold 'Em Evaluation Solver

John Hata and Noah Dunn

April 30, 2021

1 Abstract

1.1 Problem and Domain

In the world of recreational and professional gambling, Texas Hold Em' remains as one of the most prominent subsets of card-based gambling worldwide, with the WSOP (World Series of Poker) taking place every year offering millions in prizes. Although many see Texas Hold Em' as a game purely based on luck, the best players are aware that the game is far more about being able to calculate suspected odds and evaluate the standing of a given hand in the moment.

1.2 Task to be Solved

Every hand of poker revolves around every player being dealt two personal cards, which are used to complete a best hand with shared community cards. After all cards are revealed, the player with the best hand, based on a ranking system, wins the round. With this understanding, we would like to accomplish the following:

1. Determine the best hand a player can form
2. Determine the best hand their opponent can form
3. Determine who wins the hand, or who ties

1.3 Justification of ASP

ASP will enable quick hand ranking evaluation, as well as a way to use a defined rule set (poker hand rankings) to generate all possible combinations to resolve the above tasking.

2 Introduction

In order to properly encode a hand evaluation solver into ASP, a fundamental understanding of the game is necessary for predicate development. In addition, it is important to clarify some assumptions so that the solver can act with perfect knowledge of the situation, which is ideal for this solution.

2.1 Game Logistics

A game of Texas Hold 'Em can be played, theoretically, with two to twenty-two players. The variant played with two players is referred to as *Head's Up*, and all games other than *Head's Up* are referred to as n handed, where n is the number of players involved. For example, a game with five players would be called "5 handed poker". For the sake of this research, the game in question was supposed to be eight-handed poker, but due to constraints outlined later in the paper, this was changed to Head's up.

In addition, to clarify another assumption, this game will be played using only a single deck of 52 standard American playing cards. The deck will contain four suits: Clubs, Spades, Diamonds, Hearts, and 13 types of cards: Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, King, Queen, Ace. This is clearly shown in the figure 2 [1]

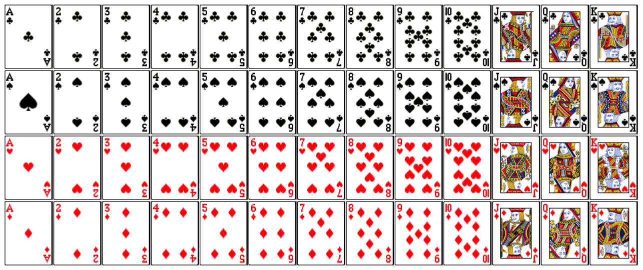

Figure 1: A set of 52 Standard American Playing Cards

Each player is dealt two cards at the beginning of each round, called *hole cards*. After a round of betting, the *flop* occurs, where one card from the deck is discarded and three cards are placed face up in the middle, called the *community cards*, from the deck. Next, the *turn* occurs, where one card from the deck is discarded and one card is placed face up in the middle from the deck to add to the community cards. Finally, the *river* occurs, where one final card from the deck is discarded and a final community card is added to the middle from the deck, leaving five cards in the middle. After the final betting round, hands are evaluated based on their strength.

[1] `http://www.milefoot.com/math/discrete/counting/cardfreq.htm`

2.2 Hand Evaluation

After the river's betting round, all players still left in the hand reveal their hole cards and construct the best possible hand of 5 cards they can build from their two hole cards and the five cards in the middle. These types of hands are shown in Figure 2.

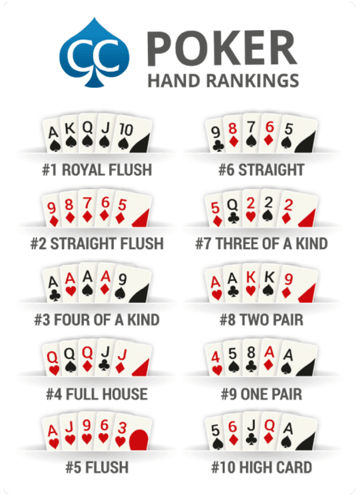

Figure 2: Poker Hands by Rank

The solver was expected to be able to take a table of eight players, and evaluate who has the best hand at the initial stage, the flop, the turn, and the river. In addition, the solver should be able to tell if two players are *chopping*, or tying a hand. Also, the solver should be able to tell the players what better hands they can form if certain cards are revealed in subsequent steps in the round (given that the players are not already at the river, where no new cards will be revealed). As discussed later, this initial scope was cut down due to the computational complexity of the problem from the initial state.

2.3 ASP Predicate Listing

This the final set of input predicates

Predicate	**Input/Output**	**Arity**	**Brief Explanation**
suit	*Input*	1	The suit value of a card (diamonds, spades, hearts, clubs)
face	*Input*	1	The face value of a card (two, three, four, five, six, seven, eight, nine, ten, jack, king, queen, ace)
rank_position	*Input*	2	How a particular card ranks against all others
rank_position_aces_high	*Input*	2	How a particular card ranks against all others with aces being the highest value
hand	*Input*	1	Facts that cover all the types of possible hands
hand	*Input*	2	A helper to ground the integers correctly
hand_pos_val	*Input*	1	Assigns a value to each hand's strength
hand_position	*Input*	3	Determines if a player can make a given hand type with the available cards
matches_face	*Input*	3	Determines if a card matches a specified face
matches_suit	*Input*	3	Determines if a card matches a suit
suit_count	*Input*	7	The frequency of a certain suit appearing in a hand
face_count	*Input*	7	The frequency of a certain face appearing in a hand
player	*Input*	1	Someone playing Poker
highest_hand	*Input*	5	Determines which of two players has the higher hand
tied_hand	*Input*	4	Determines if two players have hands of tied value
break_tie_winner	*Input*	26	If players hands are tied, breaks the tie if possible
break_tie	*Input*	13	Breaks the tie for all possible hands
matches_face	*Input*	3	Checks if a face matches a specified desired face
face_count	*Input*	7	Counts the number of times a specified face appears
matches_suit	*Input*	3	Checks if a suit matches a specified desired suit
suit_count	*Input*	7	Counts the number of times a specified suit appears
all_faces	*Input*	5	A helper to ensure that all the atoms are faces
all_suits	*Input*	5	A helper to ensure that all the atoms are suits
possible_hand	*Input*	11	A hand that a given player is able to make with his/her cards and the community cards
chosen_community_cards	*Input*	5	Three cards chosen from the total of 5 potential community cards
community_cards	*Input*	10	Five cards available in the middle to both players
find_high_card	*Input*	11	Determines the highest valued card out of 5 cards
straight_flush_evaluation	*Input*	11	Breaks ties for straights, flushes, and straight flushes
royal_flush	*Input*	10	Occurs when Ace, King, Queen, Jack, and Ten, all of the same suit are present
straight_flush	*Input*	10	Occurs when a flush (all of the same suit) and a straight (cards of increasing value) occur at the same time
four_of_a_kind	*Input*	10	Colloquially known as 'Quads', occurs when four of the same card appears
full_house	*Input*	10	Colloquially known as a 'Boat', occurs when three of one card appears, and two of another card appears
flush	*Input*	10	Occurs when there are 5 of the same suit present
straight	*Input*	10	Occurs when cards are in a row based on rank. Aces are the lowest and highest card
three_of_a_kind	*Input*	10	Colloquially known as a 'Trips', occurs when three of one card appears
two_pair	*Input*	10	Occurs when two sets of two of the same card appear
pair	*Input*	10	Pair, occurs when two of the same card appear
high_card	*Input*	10	Occurs when a player matches no other required hand
find_high_card	*Input*	11	Calculates the highest card available in the hand of five cards
game_outcome	*Output*	3	Determine who wins each game

Table 1: A Full Set of All Predicates in the Program Encoding

3 Encoding

Several rules were created in order to handle the multitude of requirements in determining what cards form what hands, how hands rank against each other, and helper predicates to limit what card sets are accepted by various rules.

3.1 Hand Determination

Ten different hands can be formed using a variety of card combinations, with different levels of specifications and restrictions for each. The hand encoding for each of the ten hands is as follows:

```
royal_flush(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
    face_count(F1, F2, F3, F4, F5, 1, ace),
    face_count(F1, F2, F3, F4, F5, 1, king),
    face_count(F1, F2, F3, F4, F5, 1, queen),
    face_count(F1, F2, F3, F4, F5, 1, jack),
    face_count(F1, F2, F3, F4, F5, 1, ten),
    flush(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5).
```

A Royal Flu

```
    straight_flush(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
        straight(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5),
        not royal_flush(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5),
        flush(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5).

    four_of_a_kind(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
        face_count(F1, F2, F3, F4, F5, 4, FACE),
        all_suits(S1, S2, S3, S4, S5).

    full_house(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
        face_count(F1, F2, F3, F4, F5, 3, FACE1),
        face_count(F1, F2, F3, F4, F5, 2, FACE2),
        FACE1 != FACE2,
        all_suits(S1, S2, S3, S4, S5).

    flush(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
        all_faces(F1, F2, F3, F4, F5),
        suit_count(S1, S2, S3, S4, S5, 5, FACE).

    straight(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
        all_faces(F1, F2, F3, F4, F5),
        all_suits(S1, S2, S3, S4, S5),
        rank_position(F1, POS1), rank_position(F2, POS2),
        rank_position(F3, POS3), rank_position(F4, POS4), rank_position(F5, POS5),
        POS5 == POS4 + 1, POS4 == POS3 + 1, POS3 == POS2 + 1, POS2 == POS1 + 1.

    three_of_a_kind(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
        face_count(F1, F2, F3, F4, F5, 3, FACE1),
        face_count(F1, F2, F3, F4, F5, 1, FACE2),
        face_count(F1, F2, F3, F4, F5, 1, FACE3),
        FACE1 != FACE2,
        FACE2 != FACE3,
        FACE1 != FACE3,
        all_suits(S1, S2, S3, S4, S5).

    two_pair(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
        face_count(F1, F2, F3, F4, F5, 2, FACE1),
        face_count(F1, F2, F3, F4, F5, 2, FACE2),
        face_count(F1, F2, F3, F4, F5, 1, FACE3),
        FACE1 != FACE2,
        FACE2 != FACE3,
        FACE1 != FACE3,
        all_suits(S1, S2, S3, S4, S5).
```

```
one_pair(S1, F1, S2, F2, S3, F3, S4, F4, S5, F5) :-
    all_suits(S1, S2, S3, S4, S5),
    face_count(F1, F2, F3, F4, F5, 2, FACE1),
    face_count(F1, F2, F3, F4, F5, 1, FACE2),
    face_count(F1, F2, F3, F4, F5, 1, FACE3),
    face_count(F1, F2, F3, F4, F5, 1, FACE4),
    FACE1 != FACE2,
    FACE2 != FACE3,
    FACE1 != FACE3,
    FACE1 != FACE4,
    FACE2 != FACE4,
    FACE3 != FACE4.
```

4 Evaluation

5 Conclusion