Generating Novice Heuristics for Post-Flop Poker

Fernando de Mesentier Silva
New York University
Game Innovation Lab
Brooklyn, NY
fernandomsilva@nyu.edu

Julian Togelius
New York University
Game Innovation Lab
Brooklyn, NY
julian@togelius.com

Frank Lantz
New York University
Game Center
Brooklyn, NY
frank.lantz@nyu.edu

Andy Nealen
New York University
Game Innovation Lab
Brooklyn, NY
nealen@nyu.edu

***Abstract*—Agents now exist that can play Texas Hold'em Poker at a very high level, and simplified versions of the game have been solved. However, this does not directly translate to learning heuristics humans can use to play the game. We address the problem of learning chains of human-learnable heuristics for playing heads-up limit Texas Hold'em, focusing on the post-flop stages of the game. By restricting the policy space to fast and frugal trees, which are sequences of if-then-else rules, we can learn such heuristics using several methods including genetic programming. This work builds on our previous work on learning such heuristic rule set for Blackjack and pre-flop Texas Hold'em, but introduces a richer language for heuristics.**

***Index Terms*—beginner heuristics, genetic algorithms, poker**

I. Introduction

If you were teaching someone to play Poker, but could only tell them a single rule for how to play, what would that rule be? That you should raise if you have a pair but fold otherwise? Now, if you could also tell them a second rule, what would that be?

Every journey starts with a first step and learning a complex skill starts with learning a simple part of that skill. Learning to drive a car usually starts with learning to shift gears, or in the US, to find the brake. This is no different in games. Playing Super Mario Bros, you will first learn to move sideways, then to jump over enemies, then to jump on top of enemies, bump into question mark blocks, shoot fireballs etc. Much later in the game you will learn complex combinations of these such as wall-jumping or finding secret passages. Every simple thing you learn makes you play the game better. These units of learning have been called "skill atoms" or "heuristics"; in this paper, we'll use the latter.

Being able to automatically subdivide a complex skill into heuristics would be very useful for being able to automatically (or semi-automatically) generate instructional sequences or tutorials for that skill. Given that we know what heuristics must be learned, and in what order, the task of figuring out how to teach those heuristics becomes much easier. But finding the heuristics necessary for a task is important for at least one other reason as well: to gauge the depth of the task. Deep tasks, for example deep games, are characterized by that mastering them requires learning a long chain of heuristics that build on each other. This can be contrasted with shallow tasks/games where there are only a small number of effective heuristics available, or there is a large number but they cannot be used together. Automatically finding the heuristics can therefore also help us estimate the depth of the task.

In this paper, we investigate methods for finding heuristics for post-flop heads-up limit Texas Hold'em Poker. This is one of the simplest versions of Poker available (chosen because it is somewhat tractable) and we are only tackling the later stages of the game. This follows on from our earlier work on finding heuristics for Blackjack and for the pre-flop phase of the same type of Poker. The basic idea is to use and search in the space of heuristic chains, and evaluate them by playing against a strong adversary.

Like in our previous work, we apply and compare greedy exhaustive search, axis-aligned search, and genetic programming, and represent heuristics in a domain-specific language meant to be able to express human-learnable rules. A major contribution in this paper with respect to our previous work is a richer language for the heuristics, able to capture late-game aspects of Poker. Other novel contributions include our approach to finding initial positions for the gameplay simulations, and our table-based reduced adversarial agent which allows fast simulation.

In the next section we describe previous work on heuristics, genetic programming, Poker AI and other topics. We then describe the particular Poker variant we are addressing, and the adversarial agent we employ. Next, we describe the heuristic language that we developed for post-flop Poker, and the details of the algorithmic approaches. In the results section, we perform comparative analysis of the quality of heuristics found through the different approaches and plot curves of policy strength relative the the length of the heuristic chain, as well as show examples of particular heuristics found.

II. Background

When making decisions, novices are not the only ones that use simple models. The theory of bounded rationality claims that the human decision making process is a factor of the amount of time available to make a decision and the amount of information available [1], [2]. Opting for simple rules-of-thumb can provide better outcome when compared to using complex algorithms because they are less prone to errors in the execution [3].

Agents that target optimum play have been able to show impressive results, in some cases being able to compete or

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

surpass professional human players. Approaches for Checkers [4], Chess [5], Go [6], Othello [7] and Poker [8] are some of the examples of such agents. Unfortunately, most of these solutions require large memory space or heavy computation that are unfeasible to be executed by human players. In order to create powerful strategies, players use heuristics to approximate the evaluation of a complex scenario. Our paper contrasts with approaches that look to maximize quality of the solution in that we constrain our heuristics to be simple to understand and execute, being careful to analyze the relationship between quality and complexity. Furthermore, we target novice players who would have been just introduced into the game.

Evolutionary approaches have also been successful at generating high performance agents. Players have been evolved for adversarial games such as Poker [9], Backgammon [10] and Lose Checkers [11]. Evolutionarhy approaches for developing strategies have also been applied to a clone of Super Mario Bros where neural nets were evolved to create controllers that play the game [12], to the solo variant of Pong [13], and to evolve controllers that exhibit general driving skills [14].

Good game-playing in poker has many different aspects to it. Bluffing, reading your opponent and calculating the risks involved in the betting are essential for high skill play [15], [16], [17]. Another important aspect of the game are what is called mixed strategies. With a mixed strategy, when observing the same game state, action A is played with probability X and action B with probability Y, as opposed to a deterministic decision. The heuristics generated in this paper are deterministic, resembling rules-of-thumb, which are known to only have a reasonable performance at the beginner level [17]. That don't pose a problem to us since beginner players are exactly the ones we wish to target with our heuristics.

The work on this paper follows our previous work on generating simple heuristics for Blackjack [18] and for the Pre-Flop round of Heads-Up Limit Texas Hold'em [19]. Using Blackjack we were able to show that it is possible to generate these heuristics and by evaluating their performance versus their complexity we approximated the skill chain [20] of the game. Blackjack was a good first step since it is a 1.5 player game, the opponent is the dealer that always plays following a known deterministic algorithm. In the Pre-Flop round paper, we showed that the same algorithms were indeed capable of generating simple heuristics on a 2-player game. In a multiplayer game your opponents are able to react and reshape their strategies during gameplay to adapt to yours, that brings the question of whether the heuristics we generated were subjected to a intransitivity relationship, meaning heuristic A beats heuristic B, heuristic B beats heuristic C, but C is able to beat A. We showed that there was in fact intransitvity between our heuristics. In this work we expand the analysis to the Post-Flop rounds of Poker, which has a richer vocabulary due to more information being available in the game state. Furthermore, we propose how a player with knowledge of the heuristics of all rounds could be able to use those to play a complete match of Heads-Up Limit Texas Hold'em.

III. Heads Up Limit Texas Hold'em Poker

Poker is a popular gambling card game. Having many variants and being played all around the world, whether it be online, casually at casinos or at a professional level. One of the most popular variations of Poker is Texas Hold'em. In this work, we will discuss how to generate novice-level simple heuristics for playing this variant of the game.

In Texas Hold'em, 2 or more players bet over multiple rounds on the best 5 card poker hand they can make out of the cards in play. The game has stochastic elements, with cards being distributed from a randomly shuffled deck, hidden information with the players holding cards in secret from each other, and as gameplay progresses more information is added, with cards that are shared by all players being revealed and actions being made. Strategies in the game usually involve bluffing, reading your opponents and assessing the risks of performing each action. Good strategies tend to have a lot of moving parts that are usually overwhelming for beginners.

A match of Texas Hold'em has 4 card rounds (in this respective order): Pre-Flop, Flop, Turn and River. Each of these rounds consists of card dealing followed by betting. In the Pre-Flop round, players are each dealt 2 cards they keep in their hand, secret from the others. In the Flop round, 3 cards are dealt face-up on the table, these are cards that are shared between all players. In both the Turn and River 1 more card is dealt to the pool of shared cards, totaling 5 face-up cards at the end. After the last round of betting, if there is more than 1 player still in the match, the showdown happens. In the showdown, the players remaining form the best 5 card poker hand they can from a total of 7 cards, their 2 cards from the Pre-Flop and the 5 cards face-up on the table. The player with the most valuable hand, following the standard hand ranking comparison, wins the total pot. On this paper, we focus on generating heuristics for playing the betting rounds in the Flop, the Turn and the River, following our previous work on the Pre-Flop round [19].

During betting, players alternate turns choosing to take 1 of the 3 possible actions. The action *Fold* has the player discarding their hand and being out of the game until the end of the match, meaning that player will not participate in the showdown. The *Check/Call* action has the player matching the largest bid made by another player so far. When using the action *Raise* players increase the highest bet to this point, turning them into the highest bidder. Once play comes back to the last player to have *Raised*, if there are multiple players remaining, the betting round ends and the match proceeds to the next round.

For the scope of this work we will be using the variant Heads-Up Limit Texas Hold'em. Heads-Up means that the matches are played by only 2 players. The variant Limit means that every time players *Raise* they can only do it by a fixed amount. Furthermore, we utilize the same settings as the Annual Computer Poker Competition for Heads-Up Limit: Only a total of 3 *Raises* can happen at Pre-Flop and 4 in the other rounds. After the limit of *Raises* has been met, the player

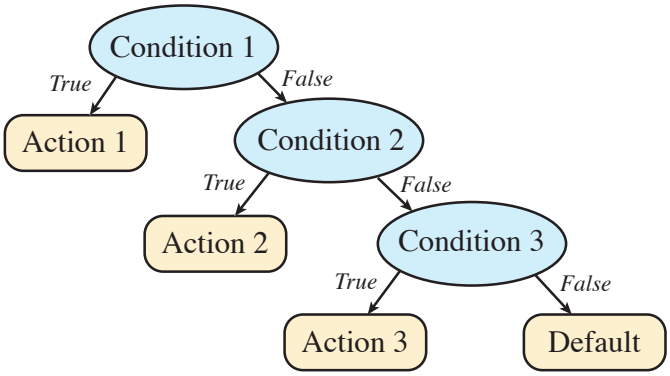

Fig. 1. Structure of a Fast and Frugal Tree (FFT). In this binary tree, each node in blue represents a condition and each node in yellow represents an action. When a condition is satisfied, the FFT returns the left child of that condition, otherwise it proceeds to next condition or, if it reached the end, returns the default action.

has to choose a different action.

The decision to use this variant of Poker is to have a simpler game to build our analysis on, and still have significant result, considering that a lot of the principles behind good heuristics in Heads Up Limit Texas Hold'em can be used to play other variants. Those are some of the reasons why this specific variant has been the focus of a lot of research, culminating in an agent that weakly solved this game [21].

IV. Heuristic Structure

Since we are looking to generate heuristics that are simple and effective, we need a proper representation. Commonly used in bounded rationality theory, Fast and Frugal Tree (FFT) is a type of binary search tree. In FFTs, nodes that have children represent conditions that inform decisions, while the leaves represent the action to be executed following the parent's outcome. Figure 1 shows the structure of FFTs. In order to make a decision we start at the condition in the root. If that condition is satisfied, we return the action on the left child of this node. If that condition fails, we proceed to the right child condition node. In case there are no more conditions to be tested, the default action (right child of the condition node of highest depth) will be returned. FFTs can also be expressed as decision lists [22], [23] and commonly have a higher performance, in practice, when compared to more complex algorithms [24].

Throughout the following sections of this paper we will represent FFTs as a chain of if-then-else statements. Their structure is as follows:

if Condition 1 **then** Action 1
else if Condition 2 **then** Action 2
else if ... **then** ...
else Default Action

We will refer to the grouping of a *condition* and the *action* resulting from successfully meeting such condition as *statement*. Each *condition* is formed by one or more boolean tests, called *clause*, that satisfy the *condition* when, and only when, are all evaluated to true. This means that a *condition* is formed by a conjunction of *clauses*. Lastly, the *action* returned by reaching the else statement is therefore the *default action*.

A. Poker Clauses

In order to generate heuristics, we first need to define a vocabulary to express them. The vocabulary is the basis of all heuristics, and determines what kind of information about the game state can be used. It is composed of several boolean functions that can be called as clauses for a condition.

When comparing to our previous work on generating heuristics for the Pre-Flop round of Heads-Up Limit Texas Hold'em, one of the main differences is the vocabulary being used. Since during the Pre-Flop players only have access to their two cards, a consequence is that the clauses rely heavily on those. For the rounds after, we no longer address the two cards in hand specifically, but rather the game it is forming together with the shared face-up cards. The only aspect of the game state that is common across all rounds is the size of the pot.

The vocabulary used in this work was built following common points of analysis discussed by players and frequently analyzed in books about Poker, such as *Play Poker Like the Pros* [15] and *The Mathematics of Poker* [16].

In the remainder of this section we describe the different boolean functions and clauses used to generate the heuristics.

1) ***Hand Rank****:* We define *Hand Rank* as the value the current player hand would have at the showdown. Since there is an order for the quality of different hands, it is possible to make statements such as: $FullHouse > Pair$ and $Flush < FourOfAKind$. Given such, we analyze *Hand Rank* in the interval: $HighestCard \leq HandRank \leq RoyalStraightFlush$. It is worth noting that this does not take into account the value of the highest card in the hand, used to break ties between hands with the same *Hand Rank*. For such, we have another boolean function.

2) ***Highest Valued Card in Hand****:* When there is a tie between 2 hands of the same *Hand Rank*, the value of the highest card that composes that hand is used to break ties. For hands of $ThreeOfAKind$ or higher it is not uncommon to play them through even when the value of the card used to form it is low. That is not the case for the lower ranks though. Considering that the bottom 3 *Hand Ranks* happen more often then all others, testing the value of the tie breaker is due necessary. Given the range of cards, the interval that can be analyzed in a clause is: $2 \leq x \leq 14$.

3) ***Total Pot****:* The size of the pot is important when playing the game since it represents the rewards for winning that match. In the heuristics we show, we represent *Total Pot* in relation to how many Big Blinds it represents. So, a test such as $TotalPot \geq 30$ is true when the pot is equal or greater than 30 Big Blinds. In Poker, Big Blind refers to the amount the last player to act in the Pre-Flop is forced to bet in the beginning of the game, to account for their favorable position.

4) ***Aggressive Opponent****:* Evaluating the behavior of the opponent can give valuable information about how a player is to proceed with their actions. Since building and analyzing

a model for the adversary is beyond the scope of this paper, we have a simplified but significant representation. We add to the vocabulary 2 boolean functions: $IsOpponentAggressive$ and $NotIsOpponentAggressive$. The first is satisfied when the last move the opponent made was *Raise*. The second just returns the inverse of the first' result.

5) ***Texture of the Board****:* By observing the face-up cards present in the board, we can analyze their texture. Texture of the Board indicates whether the shared cards shown make it likely or unlikely for players to build a good hand. For such, we create $IsBoardDry$, that is satisfied when the board is dry (unlikely to form good hands with) and $IsBoardWet$ which represents the opposite.

6) ***Outs****:* A very common and powerful strategy in Texas Hold'em is to be aware of the number of outs you have. The number of outs represents the number of cards left in the deck that if drawn to the board at later stages can improve your *Hand Rank* from what it currently is. We count the number of cards to turn Pairs into Three of a Kind, number of cards that can turn a 2 pair into a Full House and number of cards that can complete the a Flush or Straight missing 1 card. This function is not used for the River round, since no more cards will be added to the board.

V. Opponent Agent

In order to find good simple heuristics, we need to search the space of possible FFTs and evaluate their quality. To estimate the quality of a heuristic, which we will call fitness for this point on, we rely on playing a substantial amount of games and evaluating the average outcome.

Since Heads-Up Limit Texas Hold'em is a 2-player game, the fitness is calculated in relation to playing the different heuristics against the same opponent. In order for it to be feasible to play enough games in a short amount of time, we require an automated agent to play against.

For our work on Pre-flop, we were able to use the table that is part of the Nash Equilibrium used to solve the game [21]. For the post-flop rounds this table would be too large, and we don't require optimal play. Therefore we resort to a method similar to the one used to weakly solve the game. Another caveat is the time it takes for the agent to be trained. These motivate using the same methods, but under a simplified strategy, while making it more vulnerable.

The Opponent Agent we created is inspired by the work done towards building competitive AI for Heads-Up Limit Texas Hold'em [25], [26], [27] and the agent that was able to weakly solve the game [21].

This agent is built using the Counterfactual Regret Minimization (CFR) algorithm. The agent, as opposed to our heuristics which are deterministic, plays a mixed-strategy, meaning that depending on the game state it has different probabilities of making moves X, Y or Z. By starting from an equal probability for making any of the 3 moves, the algorithm then plays against itself and updates its probabilities based on the amount of regret of having performed each action. With the regret values, it updates the probabilities in relation to their positive regret (how much better it would have done if it picked another action). The algorithm has to be trained over an expressive amount of runs, relative to the size of the space that is being represented.

Since computing individual probabilities for each possible card combination is beyond the scope of this work (and tackled in Bowling *et al.* [21] and Tammelin *et al.* [28]), we opt to use a very simple card representation abstraction. For every 1 of the possible 57,344 sequences of actions, the agent classifies their *Hand Rank*, dividing hands of different rank into different buckets. This creates a probability distribution that reflect knowledge of the game and that is feasible to be trained in a small amount of time. That said, the agent has very small granularity, meaning it is vulnerable to being exploited. But, since it represents a more general play, it is unlikely to bias the heuristics generated against it. Our agent was trained using 30 cores for 7 hours, having played over 5.5 billion matches.

VI. Algorithms

Once we have an Opponent Agent, we can define our Fitness function. Such will be the average amount of money earned from playing against the agent for 400,000 matches. And the end of every match we reset each players pot and we guarantee that our heuristic will be first player through half the matches and second player for the others. Subjects with higher fitness are ranked higher in the algorithms.

We now introduce the algorithms we use to generate simple heuristics. Increasing the fitness usually reflects in an increase in complexity. So, in order to form a diverse population and be able to approximate the skill chain [20], we target creating heuristics of growing complexities, such as we did with the game Blackjack [18].

Since we are generating heuristics for the different rounds in the game, we run the algorithms separately for each case. Since the game develops over previous rounds, to avoid bias, we have the Opponent Agent play against itself until it reaches the start of the round we want to generate heuristics for. From there, we have the game be played by the heuristic until the end of the current round. Once the round is over, we deal out any cards that haven't been dealt yet, as if there was no more betting rounds in the game. After such, we proceed to the showdown to decide the outcome of this match. Once we generate heuristics for each stage of the game we will analyze how we can use these to create a strategy for playing a match from start to finish, using only heuristics found.

A. Greedy Exhaustive Search

With this algorithm, we start by initializing the population with all possible FFTs that are composed by a single condition with a single clause. They are created by assigning the boolean functions to clauses using all possible values in range and also varying all actions to be any of the 3 possible.

We proceed to evaluating the fitness of all individuals in the population, and the one with highest fitness is then the best 1-statement FFT possible. If we desire to have an individual with more statements, we repeat the process of generating all

1-statement possibilities and appending these to the end of the heuristic found on the previous step.

Since as the number of statements grow the amount of possible FFT we can build increases exponentially, it is unfeasible to exhaustively explore all possible heuristics with more than 1-statement. Even by just allowing for a statement to have more than 1 clause, the space of possible heuristics grows exponentially. As a consequence, the algorithm is very prone to finding local maximum as opposed to approaching the global optimum. Even with these caveats, it is still useful for finding the heuristics with the least amount of complexity.

B. Axis-Aligned Search

Since the space of possible heuristics is very large, performing a search for quality heuristics is computationally expensive. To avoid such, we created the Axis-Aligned search.

The method is inspired by computer graphics techniques, in particular line search, and it targets optimizing individual dimensions, one at a time, rather than the whole. The algorithm starts from a randomly generated heuristic. Then it creates a copy of the FFT, for each clause and action in the heuristic, changing only the value being compared in a clause or which action is returned. A copy is created for every possible value that can be tested for every possible clause. Copies are also created for every action. We now have a population of all these small variants of the original heuristic. We calculate the fitness for all of individuals and find the one with the highest value. The most fit now becomes the main subject. We then repeat the process, now varying all clauses and actions, except that we lock the clause/action that was changed from further mutating for the rest of the iterations. We continue with iterations until all clauses and actions are locked, or when the most fit individual of an iteration is the initial subject.

The quality of the results from this method are reliant on the random heuristic generated at the beginning. However, Axis-Aligned Search runs faster than the Genetic Algorithm, so we can have multiple runs and pick the most fit individual out of all of them. The algorithm can also be used to optimize the results found through other methods, by seeding a generated heuristic to the first iteration, instead of randomly creating one.

C. Genetic Programming

The most robust and the one most successful method at finding close to optimum heuristics for different complexities. Usually the best heuristics found come from using Genetic Programming [29], [30], [31] to search the space.

The algorithm starts with a population of 100 FFTs, of a fixed number of statements, generated at random. Then, a generation is executed by evaluating the fitness of all individuals of the population. The top 50, the elite, are selected to move to the next generation. The bottom 50 are discarded. New individuals are generated: 20 by mutating copies of elite individuals, 30 by randomly crossing over copies of the elite and then mutating the children. By maintain the top half of the population we assure that poor performing genomes do not propagate, while keeping the population diverse through having a large size for the elite and performing mutation and crossover. The algorithm performs 100 generations.

if $HandRank \geq Pair$ **then** RAISE
else CHECK/CALL

Fig. 2. Most fit least complex heuristic for Flop round. Found by Exhaustive Search, fitness is approximately 0.89.

When mutating a heuristic, the algorithm visits every clause, every constant in a clause and every action and mutates it with a 30% probability. When a clause is mutated, it is replaced with an entirely new one. When a constant is mutated, it is replace with a new constant that falls within the interval accepted by that clause. When an action is mutated, one of the other 2 actions substitutes it. Heuristics that return the same action for both evaluations of the last condition and heuristics that have a condition that always evaluates to the same value are not accepted into the population, neither are duplicates of other heuristics already in the population, and the mutation is repeated until a valid individual is created.

When crossing over 2 heuristics, a random condition or action is selected and it crosses over with the condition or action at the same depth/position in the other FFT. As it is with the mutation, with crossover creates a invalid individual or a copy of an individual already in the population, the step is repeated until a pair of valid children are returned.

Despite being the best at finding close to optimum individuals, the genetic algorithm is the most computationally expensive being roughly 25 times longer than 1 run of Axis-Aligned Search. As it is going to be discussed in the next section, the Genetic Algorithm found the majority of the most fit individuals we have for each complexity.

VII. RESULTS

In this section we showcase the results of running the algorithms described previously to generate simple heuristics for the Flop, Turn and River rounds of Heads-Up Limit Texas Hold'em. For each round we demonstrate 2 heuristics: the most fit of the lowest complexity and the most fit overall.

In order the evaluate the results found, we measure the complexity of a heuristic. For the remainder of the paper, complexity will refer to the sum of the number of clauses and number of actions (including the default action) of a heuristic. Although this evaluation does not take into account that some clauses or statements might be harder than others to parse for humans, it provides a good approximation of the amount of information that is represented.

A. Flop

Figure 2 shows the lowest complexity most fit heuristic for the Flop round. This heuristic raises if it is currently holding a hand with a pair or higher and checks/calls otherwise. Since the heuristic was trained in games that end after the Flop, holding a pair at this stage of the game can be a powerful hand, but one that can lose strength if play was to move past this round. Since there are still 2 cards left to be dealt to the

if $HandRank \geq Pair$ **then** RAISE
else if $TotalPot \leq 2$ **then** RAISE
else if $Outs \geq 11$ **then** RAISE
else CHECK/CALL

Fig. 3. Most fit heuristic for Flop round. It has a complexity of 7. Found by Genetic Programming, fitness is approximately 1.28.

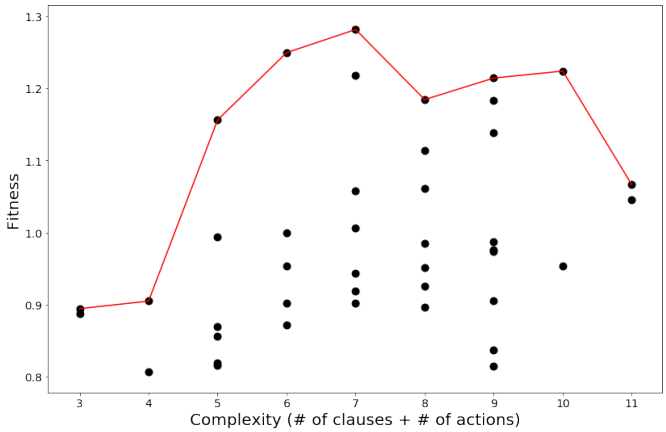

Fig. 4. Plot showcasing the relation between fitness and complexity for the heuristics found for the Flop round. The Y-axis represents fitness and X-axis represents complexity. Each dot represents 1 heuristic. The red line is an approximation of the skill chain found from the heuristics generated. Heuristics found with fitness bellow 0.8 were omitted from the graph.

board, a pair would beat other hands that have a higher reward expectation in the long run, such as a hand that is 1 card away from making a Flush.

This heuristic having positive fitness (as observed with other heuristics shown for Flop and River) indicates that it beat the opponent agent over time, but since play is cut short and later betting is skipped, it cannot be pointed as an indication that this heuristics dominates the opponents strategy.

In contrast, Figure 3 demonstrates the most fit heuristic, of complexity 7, found for the Flop round. The heuristic raises on any hand that has a pair or higher. The second statement represents a very specific scenario, the only sequence of actions that reach the Flop with total pot being 2 or lower is when all players have only checked/called in the Pre-Flop. The third statement clause checks for a state that is very advantageous to the player. If the player has 11 or more outs during the Flop, considering that at this round the deck has 45 cards and 2 more cards will be added to the board, it means that approximately 25% of the deck will improve the player's hand. Raising the pot is a strong action in these conditions.

Worth noticing is that both heuristics shown share the same first statement. This means that it could be possible to reach the best heuristic found by iterating on the best simplest heuristic. However, it is unlikely that the Greedy Exhaustive Search would have reached this result, specially considering that the second statement covers a very small part of the space of possible game states.

Figure 4 displays the relationship between fitness and complexity for the heuristics we generated. We can observe a fitness increase from complexity 3 through 7. While moving from complexity 3 to 4 grants a small gain in fitness, the greatest gain comes from reaching complexity 5. This can be explained by observing that while heuristics of complexity 3 and 4 have 1 statement, the best of complexity 5 has 1 more.

It is also worth noting that the peak of the graph is present on complexity 7, despite there being plenty of samples for complexity 8 and 9. It is unlikely that there are no heuristics of higher complexity that can outperform the current peak. It is also easy to construct heuristics of higher complexity that simply imitate the functionality of lower-complexity heuristics. It is very likely that additional search time will allow to us find heuristics with higher complexity of at least equivalent fitness. We can extend this concept to explain the sharp drop from complexity 10 to 11.

if $HandRank \geq TwoPair$ **then** RAISE
else CHECK/CALL

Fig. 5. Most fit least complex heuristic for Turn round. Found by Exhaustive Search, fitness is approximately -0.61.

if $HandRank \leq HighestCard$ **then** CHECK/CALL
else if $isBoardDry$ **then** RAISE
else if $totalPot \geq 31$ **then** CHECK/CALL
else if $HandRank \leq Pair$ **then** CHECK/CALL
else RAISE

Fig. 6. Most fit heuristic for Turn round. It has a complexity of 9. Found by Genetic Programming, fitness is approximately 0.56.

B. Turn

When observing the most fit simplest heuristic for the Turn, showcased in Figure 5, we can notice a similarity with the heuristic for the Flop. Changing from raise on pair or higher to raise on two pair or higher models a more conservative playstyle. Furthermore, it is noticeable that the fitness that before was positive is now negative. It indicates the opponent agent becomes harder to exploit further into the game.

The most fit heuristic for the Turn, represented on Figure 6, is the most granular of the ones detailed in this paper. The first statement has the players checking/calling if they reach the Turn and don't even have a pair. Next, it evaluates if the board is dry, and if so chooses to raise. This means it raises if it is holding a pair or higher and the board is dry, which puts the player in a position that seems advantageous. It proceeds to check/call if the total pot is greater or equal to 31 Big Blinds, which would indicate that the opponent has been constantly investing in their hand. In the case that the pot isn't as great, and the board is wet, it chooses to be slightly conservative and check/call if the best it has is a pair, otherwise it will raise. The heuristic has a positive fitness, meaning it is able to exploit the opponent in a more advanced stage of the game, but only by half the margin that we have reported for the Flop.

When comparing the plot for Turn, shown on Figure 7, and the Flop plot, we notice 2 main differences. The first being the

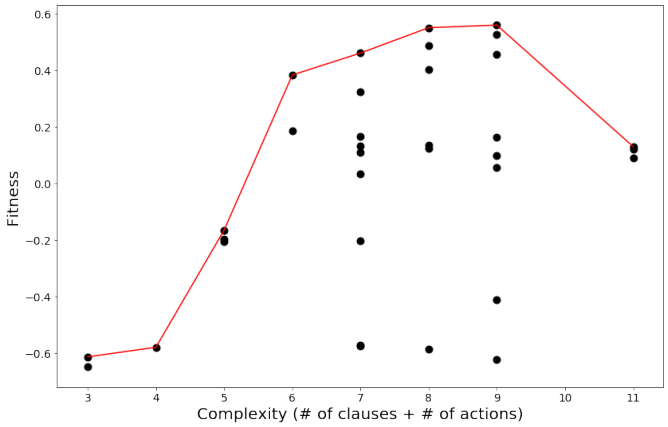

Fig. 7. Plot showcasing the relation between fitness and complexity for the heuristics found for the Turn round. The Y-axis represents fitness and X-axis represents complexity. Each dot represents 1 heuristic. The red line is an approximation of the skill chain found from the heuristics generated. Heuristics found with fitness bellow -0.7 were omitted from the graph.

if $HandRank \leq Pair$ **then** CHECK/CALL
else RAISE

Fig. 8. Most fit least complex heuristic for River round. Found by Exhaustive Search, fitness is approximately -2.27. This heuristic is analogous to the one shown in figure 2.

sharp increase that was from complexity 4 to 5 repeats, but is followed by another sharp increase from 5 to 6. The other is that there is a gain from incrementing complexity from 3 all the way to 9. It is possible that since the Turn states have more information, an extra card on the board, than the Flop it reflects on the space of viable good heuristics, making it easier for an algorithm such as Genetic Programming to find good individuals. Another observation that is worth making is how the fitness interval shifted, going from -0.6 to 0.6, when compared to the all positive interval of the Flop graph.

C. River

The most fit least complex heuristic for the last round of the game, the River, show on Figure 8, is the exact same found for the Flop and shown back on Figure 2. Despite playing exactly the same, the fitness is considerably smaller, going from 0.89 in the Flop to -2.27 in the River. This is likely due to the opponent agent having had the chance to play the game from start to finish, making it considerably more efficient than when the future betting rounds were skipped. Another detail to notice is how the best complexity 3 heuristic became more conservative only for the Turn, that is due to there is still being a lot of opportunities for improvement after the Flop and that the potential positive return of checking/calling with a pair or lower outweigh the negative return of folding at this stage.

The most fit heuristic for the River, shown on Figure 9, has 2 unique features when compared to all the others we analyzed so far: it is the only heuristic that folds and is the only heuristic to observe the opponents last action. The heuristic starts by evaluating if it the best hand it has is only a highest card and in

if $HandRank \leq HighestCard$ **AND** $OppAggressive$
then FOLD
else if $HandRank \leq TwoPair$ **then** CHECK/CALL
else if $totalPot \geq 26$ **then** FOLD
else RAISE

Fig. 9. Most fit heuristic for River round. It has a complexity of 8. Found by Genetic Programming, fitness is approximately -0.41.

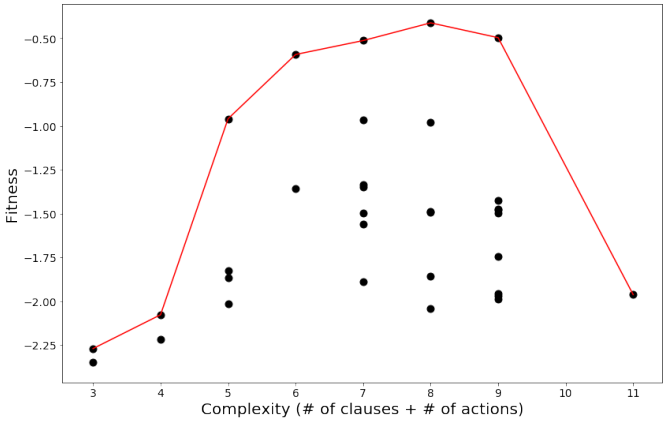

Fig. 10. Plot showcasing the relation between fitness and complexity for the heuristics found for the River round. The Y-axis represents fitness and X-axis represents complexity. Each dot represents 1 heuristic. The red line is an approximation of the skill chain found from the heuristics generated. Heuristics found with fitness bellow -2.5 were omitted from the graph.

case that is true and the opponent is being aggressive with their last move. In that case it decides to fold. It is counterintuitive that folding is only part of any heuristic when we reach the last round of the game. It proceeds to Check/Call if it is holding two pairs or lower. Lastly it once again folds if the pot is above 25 Big Blinds, otherwise it raises.

By analyzing the graph for River heuristics, shown on Figure 10, we observe that it fits somewhere in between the 2 previous. There is only 1 sharp increase, from complexity 4 to 5, but it peaks at complexity 8, with complexity 9 being considerably close in fitness. The unique feature of this graph is that the fitness interval is completely negative, meaning no matter which of the heuristics found we play, we'll lose money to the opponent agent. This corroborates our hypothesis that, when it is able to play the full game plan, from start to finish, the Opponent Agent's strategy becomes less exploitable than when skipping later betting on previous rounds.

D. Playing the full game with heuristics

With the results from our work on Pre-Flop heuristics [19] and the ones just presented, we raise the question: How well can we play the game using only our heuristics? A complete exploration and discussion of how to approach this is out of the scope of this work, but we are interested in briefly discussing the most naive approach to this: Play each round of the game with the most fit heuristic we have found for that round.

Using this method, we played 400,000 games between our best heuristics (the single best heuristic for each phase) and

our opponent agent. To our slight surprise, the set of heuristics outperformed the opponent agent, with a fitness of 3.43. Our best explanation for this is that the heuristics, which have only been trained on this particular agent, have overfit to that agent and learned to exploit it. An alternative explanation is that the representation used to create the opponent agent is too limited, and that the heuristic representation, despite its apparent simplicity, is capable of learning powerful strategies.

Even though our multiple heuristic player performs well against the Opponent Agent, it remains much more vulnerable. This is due to the fact that our heuristics are deterministic; if the opponents are able to read the heuristics play style, they will be in a very good position to exploit it.

VIII. Conclusion

In this paper we presented techniques for generating simple novice-level heuristics for the Post-Flop rounds of Heads-Up Limit Texas Hold'em. We utilized 3 different algorithms to find such heuristics, with Genetic Programming being the most successful at generating the more complex heuristics and Greedy Exhaustive Search being the most practical for generating the simplest of 1-statement heuristics.

We then proceed to analyze the most fit simplest heuristic and the most fit of heuristics found for each of the 3 Texas Hold'em rounds being discussed. We observed and compared their differences and made an attempt at creating a parallel with the thought process behind the decisions the heuristics make. We also discussed the plots of fitness x complexity for the heuristics found for each round, and the curve that approximates the skill chain [20] for each. The graphs also helped observe the differences the games' design brings to each individual round of the game.

Lastly, we proposed a naive approach to build a strategy for the full game from only the heuristics we have found. By selecting the most fit heuristic of each round, we obtained a large positive reward against the same opponent the heuristics were trained in. This raises the question of if our heuristics learned to do better than expected, or whether the heuristics found were overfit to beat that specific opponent. The answers to these question are out of the scope of this work and will instead be proposed as future work.

Acknowledgment

Authors thank the support of CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil.

References

[1] H. A. Simon, "Theories of bounded rationality," *Decision and organization*, vol. 1, no. 1, pp. 161–176, 1972.

[2] D. Kahneman, "Maps of bounded rationality: Psychology for behavioral economics," *The American economic review*, vol. 93, no. 5, pp. 1449–1475, 2003.

[3] G. Gigerenzer and D. G. Goldstein, "Reasoning the fast and frugal way: models of bounded rationality." *Psychological review*, vol. 103, no. 4, p. 650, 1996.

[4] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, "A world championship caliber checkers program," *Artificial Intelligence*, vol. 53, no. 2, pp. 273–289, 1992.

[5] M. Campbell, A. J. Hoane, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1, pp. 57–83, 2002.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[7] M. Buro, "From simple features to sophisticated evaluation functions," in *Computers and Games*. Springer, 1998, pp. 126–145.

[8] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[9] L. Barone and L. While, "An adaptive learning model for simplified poker using evolutionary algorithms," in *Congress on Evolutionary Computation, CEC 99*, vol. 1. IEEE, 1999.

[10] Y. Azaria and M. Sipper, "Gp-gammon: Genetically programming backgammon players," *Genetic Programming and Evolvable Machines*, vol. 6, no. 3, pp. 283–300, 2005.

[11] A. Benbassat and M. Sipper, "Evolving lose-checkers players using genetic programming," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 30–37.

[12] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber, "Super mario evolution," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 156–161.

[13] W. B. Langdon and R. Poll, "Evolutionary solo pong players," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3. IEEE, 2005, pp. 2621–2628.

[14] J. Togelius and S. M. Lucas, "Evolving robust and specialized car racing skills," in *IEEE Congress on Evolutionary Computation*. IEEE, 2006, pp. 1187–1194.

[15] P. Hellmuth, *Play poker like the pros*. Harper Paperbacks, 2003.

[16] B. Chen and J. Ankenman, *The mathematics of poker*. ConJelCo LLC, 2006.

[17] D. Sklansky, *The theory of poker*. Two Plus Two Publishing LLC, 1999.

[18] F. de Mesentier Silva, A. Isaksen, J. Togelius, and A. Nealen, "Generating heuristics for novice players," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[19] F. de Mesentier Silva, J. Togelius, F. Lantz, and A. Nealen, "Generating beginner heuristics for simple texas holdem," *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018.

[20] F. Lantz, A. Isaksen, A. Jaffe, A. Nealen, and J. Togelius, "Depth in strategic games," 2017.

[21] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, "Heads-up limit hold'em poker is solved," *Science*, vol. 347, no. 6218, pp. 145–149, 2015.

[22] R. L. Rivest, "Learning decision lists," *Machine learning*, vol. 2, no. 3, pp. 229–246, 1987.

[23] D. Ashlock, M. Joenks, J. R. Koza, and W. Banzhaf, "Isac lists, a different representation for program," in *Genetic Programming 1998*. Morgan Kaufmann, 1998, pp. 3–10.

[24] G. Gigerenzer, "Fast and frugal heuristics: The tools of bounded rationality," *Blackwell handbook of judgment and decision making*, pp. 62–88, 2004.

[25] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," in *Advances in neural information processing systems*, 2008, pp. 1729–1736.

[26] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron, "Approximating game-theoretic optimal strategies for full-scale poker," in *IJCAI*, 2003, pp. 661–668.

[27] A. Gilpin and T. Sandholm, "A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, no. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 1007.

[28] O. Tammelin, N. Burch, M. Johanson, and M. Bowling, "Solving heads-up limit texas hold'em," in *IJCAI*, 2015, pp. 645–652.

[29] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, no. 2, pp. 87–112, 1994.

[30] O. Kramer, *Genetic algorithm essentials*. Springer, 2017, vol. 679.

[31] N. F. McPhee, R. Poli, and W. B. Langdon, "Field guide to genetic programming," 2008.