Generating Beginner Heuristics for Simple Texas Hold'em

Fernando de Mesentier Silva
NYU Game Innovation Lab
Brooklyn, NY
fernandomsilva@nyu.edu

Julian Togelius
NYU Game Innovation Lab
Brooklyn, NY
julian@togelius.com

Frank Lantz
NYU Game Center
Brooklyn, NY
frank.lantz@nyu.edu

Andy Nealen
NYU Game Innovation Lab
Brooklyn, NY
nealen@nyu.edu

ABSTRACT

Beginner heuristics for a game are simple rules that allow for effective playing. A chain of beginner heuristics of length N is the list of N rules that play the game best. Finding beginner heuristics is useful both for teaching a novice to play the game well and for understanding the dynamics of the game. We present and compare methods for finding beginner heuristics in a simple version of Poker: Pre-Flop Heads-Up Limit Texas Hold'em. We find that genetic programming outperforms greedy-exhaustive search and axis-aligned search in terms of finding well-playing heuristic chains of given length. We also find that there is a limited amount of non-transitivity when playing beginner heuristics of different lengths against each other, suggesting that while simpler heuristics are somewhat general, the more complex seem to overfit their training set.

CCS CONCEPTS

• **Computing methodologies** → **Heuristic function construction**; • **Applied computing** → **Computer games**; • **Software and its engineering** → **Genetic programming**;

KEYWORDS

Games, Genetic programming, Empirical study

ACM Reference Format:
Fernando de Mesentier Silva, Julian Togelius, Frank Lantz, and Andy Nealen. 2018. Generating Beginner Heuristics for Simple Texas Hold'em. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3205455.3205601

1 INTRODUCTION

While developing game playing agents that approximate optimal play has been the focus of much of artificial intelligence games research, the strategies found are rarely ever applicable by humans, requiring a lot of processing power and working memory. Strategies vary according to the level of experience players have, while novices try to experiment and understand the intricacies of the mechanics, more experienced players may resort to gambles or more complex moves. In this work, we discuss how to generate game playing heuristics aimed at beginner players.

Following simple heuristics is not exclusive to novice players. Instead, the theory of bounded rationality [19, 30] reasons that humans make decisions based on the difficulty of the problem at hand, the limitation of their knowledge and the amount of time available. Applying simple heuristics in turn can be more effective, as they are less prone to cause errors in execution [15].

Players look for strategies that allow them to compete with their opponents. It is common when playing a game to find competitors of similar skill level [12]. Therefore, novice players can do well by starting from simpler heuristics and increasing their complexity as they get more experienced. The length and shape of the skill chain formed by these progressively better heuristics can indicate a game's strategic depth [24].

In this paper we will be discussing how to generate and evaluate simple heuristics for the Pre-Flop stage of Heads-Up Limit Texas Hold'em Poker. In section 2 we will contextualize our approach in relation to previous work. In the section 3 we explain the rules of the game and the Pre-Flop stage. In section 4 we discuss the agents we use as adversaries when generating our heuristics. In section 5 we discuss the heuristic structure being used. In section 6 we present the algorithms used to generate our results. In section 7 we showcase some of the heuristics we found and how they would approximate the skill chain following their fitness and complexity. In section 8 we discuss alternative approaches for evaluation following the problem of intransitivity. In section 9 we go through discussion and conclusion. Lastly, in section 10 we present the future directions for this work.

2 BACKGROUND

Agents that target optimal game-playing were shown to be able to compete on the same level or defeat expert and professional human players at games such as Chess [9], Checkers [27], Go [29] and Othello [8]. More recently computers were able to beat top human players at the game of Heads-Up No-Limit Texas Hold'em [7]. While these approaches target the optimal strategies to win, our approach focuses on discovering simple heuristics and as such we expect a trade-off between efficiency and simplicity. However, by

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205601

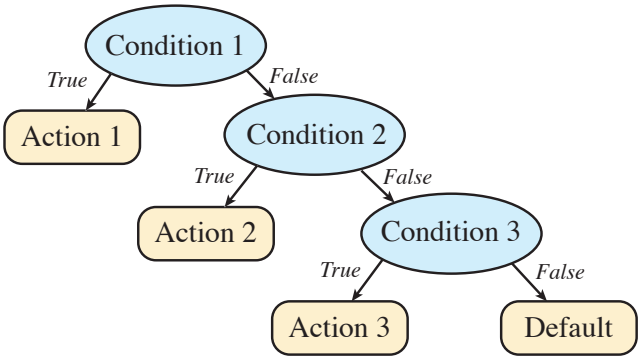

Figure 1: An example of the Fast and Frugal Tree structure. In blue are the decision nodes that represent the conditions. In yellow are the terminal nodes that represent the actions to take.

exploring the gain in quality in relation to the increase in complexity, we should in principle be able to quantitatively approximate the strategic depth of the game [24].

In order to be considered simple, heuristics need a structure that is easy to learn and read. To that effect, Fast and Frugal Tree (FFT) is simple to understand and execute, being a common structure in bounded rationality theory [14]. As shown in Figure 1, FFT is composed of a chain of nodes representing conditions that when satisfied return an action to be executed, otherwise lead to a FFT sub-tree or default action choice. The nodes can be implemented as if-then-else statements, allowing FFTs to be represented as decision lists [2, 26]. Over this work will be using FFTs to represent beginner heuristics for playing Heads-Up Limit Texas Hold'em.

We have previously introduced how to generate simple novice heuristics for the game of Blackjack [10]. In that work we detailed and discussed different algorithmic approaches and showcased how we could use the heuristics found to approximate the game's skill chain. With such, the main contribution of this work is tackling the Pre-Flop stage of the game Heads-Up Limit Texas Hold'em Poker and exposing the differences in approach and the influence it has on the method of evaluation. Different than Blackjack, considered a 1.5 player game, where the players are playing against a previously known deterministic agent (the dealer), in this form of Poker two autonomous players go head-to-head, each playing according to their own strategy, undisclosed to their opponent previously. The fact that the adversary is now an intelligent agent, that could change strategies between games, requires a change in methodology in how we approach comparing the quality of heuristics. A two-player game also introduces the problem of intransitivity: If I have heuristic A that wins against heuristic B and heuristic B wins against heuristic C, that does not guarantee that A will win against C.

Evolutionary algorithms have been able to learn effective game-playing strategies for games. For example, there are evolutionary approaches to the Chess endgame [16], Backgammon [3], Lose Checkers [5], Pong [23]; neuroevolution for Super Mario Bros [34]; or evolved agents for Core War [36], Robocode [28], Pac-Man [21]. Evolution strategies were also used to generate general driving controllers [1] and Blackjack strategies [13, 18, 20, 25]. Poker has also been a subject of evolutionary approaches, with the target being to create agents that could maximize winnings [4]. Our approach is focused on creating a heuristic that can be read and understood by novice players and to achieve such there is a cost to their fitness, so we have to find a balance between quality and complexity.

Our approach is related to the one used by Tsang et. al [35]. They utilize genetic programming to build decision trees used as a forecasting tool for investments. The authors intended to create a system that could generate, with the help of historical data, human-readable trees that could then be accepted or rejected by the users. Although our approach doesn't expect human feedback towards the results of our generation, it is a hard constraint that humans can read and be able to execute are proposed solutions.

The game Heads-Up Limit Texas Hold'em has been weakly solved recently [6, 32]. In their work, Bowling et. al used a variant of counterfactual regret minimization, computing approximations of a Nash equilibrium type of solution. The solution presents input to common strategy discussions about what would be optimal play. The agent employs a mixed strategy: It has different probabilities for taking the actions, given the current state of the game. The heuristics we present on this paper provide deterministic play, as opposed to mixed-strategies, and despite being commonly stated that Rules of Thumb do not go above beginner level [31], this is exactly the skill level of the players we target.

3 HEADS-UP LIMIT TEXAS HOLD'EM POKER

Texas Hold'em is one of the most popular variations of Poker. It is commonly played at casinos and online. In Texas Hold'em, each player has their own hand of 2 cards, kept hidden from others, and share a common pool of 5 cards face-up on the table. Players are betting on the best 5-card poker hand they can build from the 7 cards available to them.

A hand in Texas Hold'em is played over 4 stages, each adding more information to the gamestate. At each stage, players take turns choosing 1 of 3 possible actions: Raise, Call/Check, Fold. Raise sees the player increase the amount of money being bet by matching and increasing the current highest bet. In a Call/Check the player matches the size of the current highest bet. Deciding to Fold has players abdicating from all money they bet up to this point and no longer being a contender to win the round. At the first stage, the Pre-Flop, players take actions knowing only their two cards. On the second stage, the Flop, players take actions after the first 3 face-up cards are revealed. On the third stage, the Turn, the forth card is added to the common card pool. The last stage, the River, reveals the last face-up and ends once no more players are able to take actions. At the end, players that have not folded compare their poker hands and the highest hand takes the money pot.

Before the round starts, a player assumes the role of dealer, dealing out cards to all players. The role of dealer is passed on to the next player when new round starts. The player to the left of the dealer has to bet a fixed amount of money, the small blind. Furthermore, the next player has to bet twice that amount, the big blind, before the round starts.

In this work we work with one variant of Texas Hold'em. Limit Texas Hold'em refers to a restriction on the betting, more specific the Raise action. Every Raise increases the current highest bet by

the amount of 1 big blind. Furthermore, Heads-Up refers to a 2-player game of Poker. In this work we will then be generating simple heuristics for playing the Pre-Flop stage of Heads-Up Limit Texas Hold'em Poker.

4 ADVERSARY AGENTS

In order to generate heuristics, we need to have a method to determine their fitness. Since the game revolves around betting, we use the average amount of money earned over the course of several games as the fitness for an individual. Since we are only focused in playing only the Pre-Flop round, we simulate the rest of the game without any bets occurring, comparing the hands of the players after dealing all remaining cards. In order to play these games, the heuristics need an opponent. Due to the stochastic nature of Poker mechanics, we play 400,000 hands with each individual to calculate their fitness. For this purpose we have 2 different agents and have the heuristics play 200,000 hands against each, reseting the money earned at the end of each and alternating the player that acts first.

The first agent is inspired by the beginner Pre-Flop strategy described in the book *Play Poker Like the Pros* [17]. In it, Phil Hellmuth, a professional poker player, suggests playing only the hands described as the top ten hands, choosing to Fold in all other situations. The top ten hands presented are, from best to worst: A-A, K-K, Q-Q, A-K, J-J, 10-10, 9-9, 8-8, A-Q and 7-7. The author describes this strategy as suitable to keep the players "in the game" while they learn the intricacies required for higher level play.

The second agent was built based on Cepheus, the agent created for tackling the problem of solving Heads-Up Limit Texas Hold'em [6]. It represents a fraction of one possible Cepheus strategy. With the data of one Nash Equilibrium generated by Cepheus for Pre-Flop play, we attempted to make an agent that would recreate this strategy. The data provides the probabilities of each possible action based on the 2 cards the agent has and the sequence of actions in the game so far. With the probabilities mapped, we use a random number generator to return a number n, using a uniform distribution, such that $0.0 \leq n \leq 1.0$. This n combined with the action probabilities then defines the action the agent will take.

5 HEURISTIC STRUCTURE

As mentioned before, all our heuristics are deterministic. The heuristics we generate are represented as fast and frugal trees, or in this case decision lists representing chains of if-then-else statements. A heuristic could have any number of statements, but more statements result in a more complex heuristic. Each statement in turn is formed by a condition and a resulting action, with the else statement at the end returning the default action, the one used when all the conditions before evaluate to false. Conditions in turn are formed by a single clause or a conjunction of multiple clauses.

To form heuristics for Heads-Up Limit Texas Hold'em the clauses can test different elements of the gamestate. The 2 cards in the player's hand are addressed as *highestCard* and *lowestCard*, which represent the card of highest and lowest value respectively. Considering the face cards Ace, King, Queen and Jack to be of value 14, 13, 12 and 11, *highestCard* and *lowestCard* can be compared on the interval $2 \leq x \leq 14$.

Two boolean checks can also be clauses: *hasPair* and *isSameSuit*. The first, *hasPair*, is true when both cards in the player's hand have the exact same value. The second, *isSameSuit*, in turn evaluates if both cards are of the same suit. The negative of each boolean check, not hasPair and not isSameSuit, are also added as possible clauses.

The test *cardDifference* evaluates the gap between the cards in the player's hand. The gap is analogous to the result of subtracting the values of the highest card by the lowest card. The card gap can then be in the interval $0 \leq x \leq 12$.

Lastly *totalPot* represents the total money resulting from summing the amounts that each player has bet so far. In terms of clauses, we represent the total pot in units of number of big blinds, so a pot of 5 is equivalent to 5 big blinds and a pot of 10 represents 10 big blinds. In our simulations of the game, each player has a total of 50 big blinds available for betting at the start of new hand.

Our heuristics are then formed according to the following structure:

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

With each condition being formed by one or more clauses from the ones we listed above. The actions for our heuristics can have 1 of 3 values, representing the actions that exist in the game: Raise, Check/Call and Fold.

We also evaluate each heuristic in a metric of complexity. We calculate complexity by counting the number of actions plus the number of clauses (considering all conditions). Although this number doesn't take into account that, for humans, one clause might be easier to understand or test for than another, this simple calculation gives us a clean simple way to compare heuristics.

6 ALGORITHMS

In order to generate the different heuristics we resort to a selection of search algorithms which we discuss in this section. Each technique we present in here has trade-offs, and employing all of them results in a more robust set of heuristics. We first introduced these techniques when generating simple heuristics for Blackjack, with different primitives and fitness function.

6.1 Greedy-Exhaustive Search

Ideally when teaching beginners good strategies we would like to start with the most basic guideline and gradually, as they improve, introduce extra steps that complement the original, resulting in stronger play. With that goal in mind we turn to this approach.

When generating heuristics with Greedy-Exhaustive Search, we start from the simplest we can have, the best possible 1 statement heuristic. In order to find such, we generate all possible heuristics we can have with only 1 condition, by conjugating all variations of our clauses with all variations of our actions. We then proceed to calculating the fitness of each of those possible candidates and the one with the highest score is returned as the result of the search.

In order to increase the complexity of the heuristic, aiming at also increasing the fitness, we fix the statement and default action

we previously found and proceed to repeating the process for generating following statements, which are appended after the ones we already found.

Although this algorithm can possible increase the quality of the heuristic with the addition of new statements, it is also very prone to finding local maximums. Due to this effect, it is not uncommon for the heuristic to stop improving after a certain step, while also having lower fitness than another heuristic of the same complexity found with another method. Performing a non-greedy exhaustive approach to generate more than 1 statement at a time becomes unfeasible very fast due to the space growing exponentially with the increase in number of statements.

6.2 Axis-aligned Search

In order to reduce the search space for heuristics we devised a technique called Axis-aligned Search. The concept is to exhaustively search along a fixed axis so as to largely reduce the search space. Each clause and each action present in a tree is considered an axis that we perform the search on.

The algorithm starts by generating a heuristic with random clauses and statements. For each clause we generate another N candidate copies of the current heuristic, each diverging from all the others only in the value being compared in the clause. We then have N variants for the N possible values of a clause (N varying depending on the clause). The same concept is repeated for each action present in the heuristic. We then evaluate the fitness of all candidates. We pick the most fit for the current heuristic and fix the axis that was changed to generate it. We then proceed to the next iteration where new candidates will be generated from all axes that haven't been fixed yet. We repeat this loop until we fixed all axis or an iteration returned the same heuristic it had at the start of it.

Axis-aligned Search, due to the randomness element of the starting step, has a high variance in the quality of individuals it finds. However, since it is a much more inexpensive algorithm than genetic programming, which we will discuss next, we are able to run a significant amount of trials in a reasonable amount of time and then choose to keep the most fit heuristics found.

6.3 Genetic Programming

Finally, in order to look for the fittest heuristic for a given complexity we use genetic programming [22, 33]. We restrain our search to heuristics with a fixed number of statements and run the algorithm multiple times, with different numbers, to generate heuristics of different complexities.

To start the algorithm we have a population of randomly generated FFTs with a set number of conditions. The population is generated to guarantee that no 2 individuals are the same. In every generation, the most fit 50% of the population creates the elite. We then create new candidates by mutating copies of the top 20% individuals of the population. Furthermore, we crossover members of the elite, picked at random, until we generate a total of new individuals equal to 30% of original population size, always keeping the original individuals involved in the crossover. The resulting children are then submitted to mutation. At the end of a generation we then have a population of elite + mutated elite + mutated crossover children.

The mutation algorithm traverses every node of the tree, choosing to mutate them based on the mutation probability. When a node is selected for mutation; if the node is a clause it will change it to an entirely new one; if the node is a terminal, which is not an action, it will be substituted for a terminal of different value; if the node is an action, it will be replaced with one of the 2 other actions. Mutating the 2 actions on the deepest level of the tree, the default action and the action resulting from the last condition, can lead to both being the same action, which would render the last condition useless. To avoid such, we repeat the mutation step when in this situation.

The crossover operator performs a one point crossover. A subtree is randomly selected for crossover on one individual. Then, the subtree on the same level and of the same depth on the other individual is used to complete the crossover. The two child individuals generated are then returned as the result of the crossover, without destroying their parents.

We run our genetic algorithm with a population of 200, for 100 generations. The fitness function used is the one described in the adversary agents section. We set a probability of 30% when mutating the elite and of 10% when mutating the crossover children.

The genetic algorithm is more expensive than the axis-aligned search, but the results across runs have much smaller variance. Using a different approach for the genetic algorithm than what we previously did for Blackjack [10], it was able to generate the best results for most of the heuristics complexities when compared to the axis-aligned search. We believe that this is also due to Texas Hold'em Poker requiring a larger vocabulary than Blackjack, which increased the variance of axis-aligned, which was previously more successful in finding the best fit individuals.

7 RESULTS

Our objective is to generate simple heuristics for Heads-Up Limit Texas Hold'em. With the algorithms presented on the previous section, we proceed to generate heuristics of various number of statements and with different complexities. We attempt to find the best heuristic of each complexity, which can take multiple runs of one or more of the algorithms, so we can approximate the skill chain of the game.

To generate the 1-statement heuristics we use greedy-exhaustive search. Figure 2 shows the most fit heuristic found for complexity 3. This represents the most fit simplest heuristic our methods can generate. It represents a "safer" play style, only choosing to raise in hands where both cards have value above 10. The heuristic loses money over time, having a fitness of approximately -0.228.

if $lowestCard \leq 10$ **then** Check/Call
else Raise

Figure 2: The best simplest heuristic we can generate. This heuristic was found with greedy-exhaustive search. It has complexity 3. Fitness is -0.227784375.

From the previous heuristic, we proceeded to use Greedy Exhaustive Search to generate the second statement. In parallel, we had the genetic programming algorithm search for 2-statement heuristics. The best heuristic found with GA proved to outperform the fitness

of the Greedy-Exhaustive 2-statement heuristic by over 0.5. This showcases the problem of local maximum convergence present in the Greedy-Exhaustive technique. Figure 3 shows the heuristic found by GA. Although this heuristics raises on more hands than the previous, it only raise very early in the game, choosing only to do so only while there are at most 2 big blinds on the pot. Despite that, the heuristic displays a significant gain in fitness, no longer losing money, winning an average of 0.5.

if $highestCard \leq 10$ **then** Check/Call
else if $TotalPot \leq 2$ **then** Raise
else Check/Call

Figure 3: The best 2-statement heuristic of this complexity, found using genetic programming. Complexity is 5. Fitness is 0.50010625.

When analyzing the most fit 3-statement heuristic, we note that, once again, the best individual is not derived from the heuristic found in the previous step. Figure 4 shows the result in case. This heuristic does not raise hands where the lowest value card is bellow 8, unless it also has an Ace, King or Queen. Even then, it only raise at the very early game, similar to the previous heuristic, except when it has a pair. The heuristic has the player continuously raising a hand that has a pair of 8 or higher. This decision to play high pairs aggressively follows a similar set of mind as the beginner Pre-Flop strategy suggested by Phil Hellmuth on his book [17]. This heuristic shows considerable improvement over the previous, scoring a fitness of approximately 0.778 with complexity of 8.

if $lowestCard \leq 7$ **and** $highestCard \leq 11$ **then** Check/Call
else if $TotalPot \leq 2$ **then** Raise
else if $hasPair$ **then** Raise
else Check/Call

Figure 4: The best 3-statement heuristic found using genetic programming. Complexity is 8. Fitness is 0.777696875.

Going against the trend set by the previous examples, the best 4-statement heuristic we found is directly derived from the best 3-statement candidate discussed before. Both heuristics were found on independent runs of GA. Figure 5 shows the representation of the 4-statement individual found. This heuristic of complexity 10 adds an extra statement that has the player fold when the adversary is being aggressive and raising constantly, given that the hand does not fit the conditions that precede it. This could be an indication of the heuristic overfitting to the opponents it was trained against as it realizes that if they kept raising under does circumstances, they were likely to have a better hand. This heuristic has a complexity of 10, which is 2 more than the previous heuristic we discussed, meanwhile its fitness is approximately 0.74, which represents a loss of about 0.03 over the previous.

7.1 Comparison of Heuristics and Skill Chain

Once we had generated a diverse enough group of heuristics it was possible to plot those heuristics in terms of their complexity and

if $lowestCard \leq 7$ **and** $highestCard \leq 11$ **then** Check/Call
else if $TotalPot \leq 2$ **then** Raise
else if $hasPair$ **then** Raise
else if $TotalPot \geq 6$ **then** Fold
else Check/Call

Figure 5: The best 4-statement heuristic found with genetic programming. Complexity is 10. Fitness is 0.743465625.

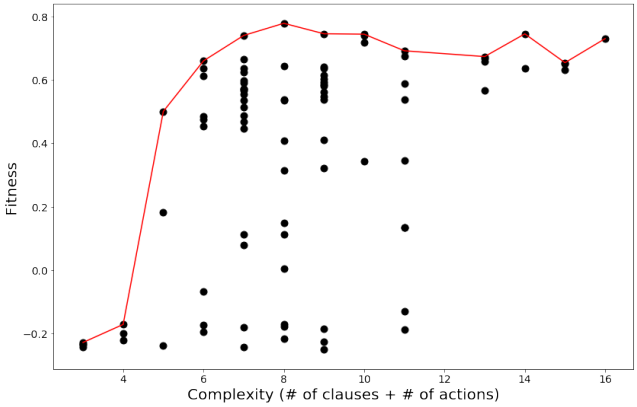

Figure 6: Chart comparing different heuristics. Each dot represents a different heuristic generated. The X-axis represents the complexity of the heuristic, meanwhile the Y-axis shows the fitness. The red curve that connects the most fit individual of every complexity is a tentative candidate for an approximation of the skill chain.

fitness in order to approximate the skill chain curve for the Pre-flop stage of Limit Texas Hold'em. Figure 6 shows this comparison.

Following the shape of the curve, we observe a significant gain in fitness from complexity 4 to 5 and above. One of the main differences between these heuristics is the number of statements: while heuristics of complexity 3 and 4 are 1-statement long, all our heuristics above that complexity have 2 or more statements. Furthermore we notice that the gain in fitness gradually reduces as heuristics get more complex, until it eventually converges, showcasing little to no gain from adding clauses or statements. This curve resembles the one we found for Blackjack [10]. This would indicate that, in the Pre-Flop stage of Heads-Up Limit Texas Hold'em, there are small steps a beginner can make to improve their game considerably. Another point to notice is the slight drop in performance and small amount of samples for higher complexity heuristics. Due to the increase in computation cost, an inferior number of runs was made for heuristics with more than 5 statements.

This graph would be sufficient for itself, had the game possessed a deterministic opponent, such as the dealer on Blackjack, but since this is a 2-player game, a couple questions need to be raised. The first one being if there is a intransitivity problem. Since each player chooses their own strategies, it is possible to have a relationship where heuristic A is dominant against heuristic B and B is dominant against heuristic C, but C is dominant against A. If we notice that intransitivity in fact occurs with the heuristics we found, a

consequent question is raised: How can we evaluate the different heuristics in order to compare their fitness?

8 ALTERNATIVE EVALUATION METRICS

In order to investigate the existence of non-transitivity, we chose a group of heuristics, from the ones we found, and played them against every other one in the group. For this experiment we chose the most fit heuristic of every complexity. We can then look to notice if heuristics that are most fit when playing against our automated agents still have the upper-hand when going head-to-head versus one another. Figure 7 displays the results of the match-up of different heuristics.

The columns and rows of the table are labeled for the complexity of the heuristic they represent. The table shows the average amount of money the row heuristic earned when playing the column heuristic. By analyzing the table, we can see evidence of non-transitive relationship between heuristics. Looking at heuristics 8, 9 and 10 we observe that 8 beats 9, 9 beats 10 and 10 is the only heuristic capable of earning money versus 8. The relationship repeats itself for 7, 13 and 14, with 7 triumphing over 13 by a reasonable margin, 13 beating 14 with a small margin, and 14 having the best performance of a heuristic against 7.

The table also reveals peculiarities about the match-ups, if we analyze it in relation to the heuristic comparison chart of Figure 6. Having 14 lose to 9, 10, 11, 13 and 15 despite showing better fitness. Also, the case of 11, 13 and 15 that have much higher value when they lose, in most cases, than other heuristics of complexity 6 or above. These facts lead us to the hypothesis that higher complexity heuristics might be overfitting to our automated agents. Having more than 5 statements, as is the case for all the heuristics above complexity 11 we found, may have tailored their playstyle to explore some weakness specific to these agents, instead of making more generally applicable decisions.

In order to validate the quality of our heuristics and test the hypothesis of overfitting raised from analyzing the match-up table, we propose an alternative evaluation method.

8.1 Heuristic Set Evaluation

To evaluate the generality of the play-style of our heuristics, we chose a small subset of all heuristic found this far as a test group. We then proceeded to repeat our original analysis comparing individuals, but now having fitness as the average earnings from playing all members of the test group.

For the subset of heuristics that formed the test group, we want to create a population that is diverse enough while also having the highest possible fitness. The first step, in order to generate diversity, is to create a method to measure the distance between 2 individuals. We start by mapping, for each heuristic, the action chosen for every possible hand, for all possible values for the total pot. We then compute the hamming distance between these mappings.

With a method to measure distance, we compute, for each heuristic, their distance to a heuristic that returns Raise for every move and to a heuristic that returns Check/Call for every move. Then, in order to look for individuals that are representative of a different diversities, we use k-means to calculate the center of the different clusters found. Once we have such, we allocate the points in each cluster, by selecting the one with the center they are the closest to. We then pick the most fit heuristic of each cluster to be their representative. Figure 8 shows the distribution of heuristics in the space, their fitness and the center of each cluster found with the k-means algorithm for k=8.

Complexity	Fitness	Cluster	Best of Complexity
5	-0.2656	7	No
7	0.6648	2	No
8	-0.1771	4	No
8	0.7777	6	Yes
10	0.7434	3	Yes
11	0.6912	5	Yes
13	0.6583	8	No
14	0.7444	1	Yes

Table 1: Table showing information about the heuristics that compose the test group. Each row represents one heuristic. The Complexity column indicates the complexity of the heuristic. The Fitness column is the fitness when playing the automated agents. The Cluster column refers to which cluster the heuristic belongs to, according to the labels on Figure 8. The Best of Complexity column represents whether that heuristic is the most fit of its complexity.

The evaluation group was then formed by 8 heuristics across 7 different complexities. Table 1 shows information about the heuristics in the group. The group has 2 individuals with negative fitness, which represent the clusters on the top-left and top-right of Figure 8. Out of all heuristics, half are the most fit of their own complexity. This data leads us to believe that these heuristics form a heterogeneous group, fitting with we were looking for.

After deciding the test group, we defined a new fitness function. We calculate the fitness of an individual by having it play 800,000 matches against each of the 8 heuristics on our evaluation set. We then return the average money gained across all matches as the fitness. We then proceeded to repeat our analysis of Complexity versus Fitness. Figure 9 shows this comparison.

By comparing the resulting graph in relation to the previous evaluation we quickly take note of the discrepancy in shape. While the previous curve was smoother, this presents a drop when moving from complexity 7 forward, followed by another increase followed by another sharp drop. We also proceeded to form a line, colored blue in Figure 9, that connects the heuristics that used to be most fit, for each complexity. We notice that in many cases, specially for the higher complexity individuals, not only are they no longer the best, but also present a sharp drop in comparison to other heuristics that they once bested. This graph give us an indication that overfitting is in fact happening as the complexity grows. Specifically in the case of complexity 11 and 13, we can observe that the heuristic that once was the most fit, has now been surpassed by another candidate by a significant margin.

9 DISCUSSION AND CONCLUSION

In this paper, we have shown and discussed techniques to generate and evaluate simple heuristics for playing Pre-Flop Heads-Up

	3	4	5	6	7	8	9	10	11	13	14	15	16	total
3	0	0.03495	-0.9247	-1.0214	-1.10823	-1.05442	-1.189	-1.13552	-1.08248	-0.996175	-1.14777	-1.14098	-1.1062	-11.871925
4	-0.03495	0	-0.777475	-1.06937	-1.25137	-1.18738	-1.1722	-1.27115	-0.59295	-0.4787	-1.13573	-0.616275	-1.13223	-10.719775
5	0.9247	0.777475	0	-0.191175	-0.329925	-0.174525	-0.276325	-0.28025	-0.35595	-0.31375	-0.288475	-0.3543	-0.253075	-1.115575
6	1.0214	1.06937	0.191175	0	-0.060025	-0.081	-0.0018	-0.0466	-0.157125	-0.091075	-0.0881	-0.103275	-0.117225	1.535725
7	1.10823	1.25137	0.329925	0.060025	0	-0.015	0.03005	0.06445	0.572575	0.6378	-0.040775	0.479025	0.052425	4.5301
8	1.05442	1.18738	0.174525	0.081	0.015	0	0.06025	-0.02605	0.456375	0.46545	0.046125	0.473	0.0128	4.000275
9	1.189	1.1722	0.276325	0.0018	-0.03005	-0.06025	0	0.014025	0.192075	0.177025	0.06475	0.184175	0.001775	3.18285
10	1.13552	1.27115	0.28025	0.0466	-0.06445	0.02605	-0.014025	0	0.5358	0.609975	0.05085	0.494725	0.016375	4.388825
11	1.08248	0.59295	0.35595	0.157125	-0.572575	-0.456375	-0.192075	-0.5358	0	-0.1038	0.0259	-0.058925	-0.01675	0.2781
13	0.996175	0.4787	0.31375	0.091075	-0.6378	-0.46545	-0.177025	-0.609975	0.1038	0	0.111775	-0.02525	0.0689	0.248675
14	1.14777	1.13573	0.288475	0.0881	0.040775	-0.046125	-0.06475	-0.05085	-0.0259	-0.111775	0	-0.081575	0.047125	2.367
15	1.14098	0.616275	0.3543	0.103275	-0.479025	-0.473	-0.184175	-0.494725	0.058925	0.02525	0.081575	0	-0.004325	0.745325
16	1.1062	1.13223	0.253075	0.117225	-0.052425	-0.0128	-0.001775	-0.016375	0.01675	-0.0689	-0.047125	0.004325	0	2.4304

Figure 7: Table showing the results of the Match-Up of different heuristics played over 800,000 games. Each row represents the most fit heuristic of that complexity. Each value is the fitness of playing the row heuristic position versus the column heuristic. The diagonal is not computed (the heuristic playing against itself). Cells are color coded: the greener a cell is higher the value, the redder a cell is lower the value. The last column indicates the total from summing the values of the row.

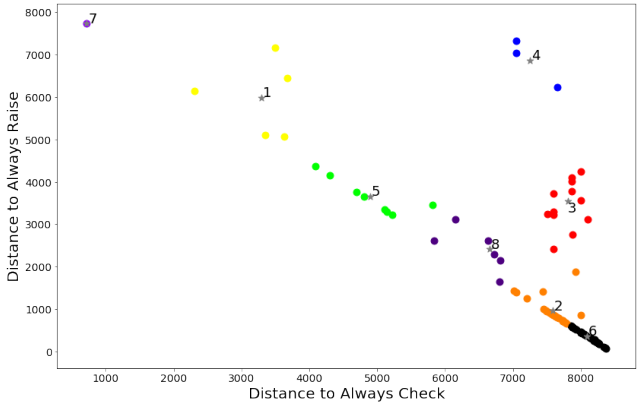

Figure 8: Distance comparison between heuristics. The X-axis represents the hamming distance to a heuristic that uses Check/Call in every single scenario. The Y-axis represents the distance to a heuristic that always uses Raise. The color of the dots indicate their cluster. Each gray stars represents the center of one of the k-means clusters for k=8.

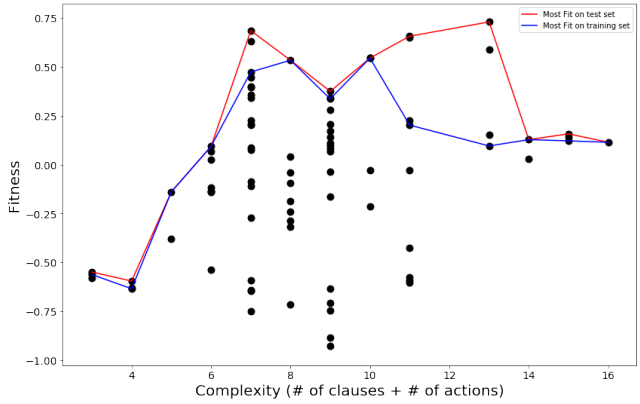

Figure 9: Comparison of complexity versus fitness for all heuristics we found. Fitness is calculated using the test group of heuristics. The red line connects the most fit individuals. The blue line connects what were previously the most fit individuals (the comparison in Figure 6).

Limit Texas Hold'em. Our heuristics are meant to be simple, to be usable by novice players, and with are therefore represented as Fast and Frugal Trees. The main differences between this approach compared and the one we previously presented for Blackjack [10] is the introduction of a new vocabulary and a set of approaches to analyze the heuristics for a 2-player game.

We reintroduced the 3 different algorithms used for generating the heuristics. Greedy-Exhaustive search is efficient at finding very simple heuristics, but is likely to converge to a local maximum. Axis-aligned search has short runtime, but high variance due to the random element at the initial step. Genetic Programming finds good results more consistently, but has at a higher computational cost. Most of the higher quality heuristics, independent of their complexity, were found using Genetic Programming. This finding diverges from what we observed when searching for Blackjack heuristics, Axis-aligned Search was more successful in generating quality heuristics in that game. This can be a consequence of Pre-Flop Heads-Up Limit Texas Hold'em having a larger space of possible heuristics, in comparison to Blackjack.

Playing against an opponent that can change and adapt their approach as the game progresses makes 2-player games raise new challenges, particularly the intransitive nature of strategies. This makes the evaluation and comparison methods we used to rank heuristics for Blackjack be much less effective. To remedy this, we introduced a methodology to verify and analyze the impact of intransitivity in the heuristics found. By creating a test set, formed by selecting heuristics from the ones found in order to have a heterogeneous group, we were able to create a new evaluation metric. When comparing the results of the original method with the one from the test set, we observed strong indicators of large heuristics (with complexity larger than 7) being overfitted to their training set. Given these results, although we have yet to find a reliable approximation of the complete skill chain of the game, we made significant steps towards discovering it, and with such a method to estimate the strategic depth.

10 FUTURE WORK

Having only explored the Pre-Flop stage of the game, the next logical step is to evaluate how our techniques perform on the other stages of the game. Each stage adds more information to the gamestate and requires a different vocabulary to generate the clauses from. Another future challenge is to explore regular Limit Texas Hold'em, having more than 2 players involved in the game. Furthermore, we would like to be able to represent mixed strategies, opening the possibility for better heuristics.

We believe that generating simple heuristics can be an asset to explore solutions for games we don't know good strategies for, such as a game under development, enabling solutions such as partially automated playtesting [11]. Another possible application is to use the data from the heuristics found to build automated agents that could adapt their play-style during the game.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Alexandros Agapitos, Julian Togelius, and Simon Mark Lucas. 2007. Evolving controllers for simulated car racing using object oriented genetic programming. In *Genetic and evolutionary computation*. ACM, 1543–1550.
[2] Dan Ashlock, Mark Joenks, John R Koza, and Wolfgang Banzhaf. 1998. ISAc Lists, A Different Representation for Program. In *Genetic Programming 1998*. Morgan Kaufmann, 3–10.
[3] Yaniv Azaria and Moshe Sipper. 2005. GP-gammon: Genetically programming backgammon players. *Genetic Programming and Evolvable Machines* 6, 3 (2005), 283–300.
[4] Luigi Barone and Lyndon While. 1999. An adaptive learning model for simplified poker using evolutionary algorithms. In *Congress on Evolutionary Computation, CEC 99*, Vol. 1. IEEE.
[5] Amit Benbassat and Moshe Sipper. 2010. Evolving lose-checkers players using genetic programming. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 30–37.
[6] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. 2015. Heads-up limit Hold'em poker is solved. *Science* 347, 6218 (2015), 145–149.
[7] Noam Brown and Tuomas Sandholm. 2017. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* (2017), eaao1733.
[8] Michael Buro. 1998. From simple features to sophisticated evaluation functions. In *Computers and Games*. Springer, 126–145.
[9] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. 2002. Deep blue. *Artificial intelligence* 134, 1 (2002), 57–83.
[10] Fernando de Mesentier Silva, Aaron Isaksen, Julian Togelius, and Andy Nealen. 2016. Generating heuristics for novice players. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 1–8.
[11] Fernando de Mesentier Silva, Scott Lee, Julian Togelius, and Andy Nealen. 2017. AI-based playtesting of contemporary board games. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*. ACM, 13.
[12] George Skaff Elias, Richard Garfield, K Robert Gutschera, and Peter Whitley. 2012. *Characteristics of games*. MIT Press.
[13] David B Fogel. 2004. Evolving strategies in blackjack. In *Congress on Evolutionary Computation, CEC2004*, Vol. 2. 1427–1434.
[14] Gerd Gigerenzer. 2004. Fast and frugal heuristics: The tools of bounded rationality. *Blackwell handbook of judgment and decision making* (2004), 62–88.
[15] Gerd Gigerenzer and Daniel G Goldstein. 1996. Reasoning the fast and frugal way: models of bounded rationality. *Psychological review* 103, 4 (1996), 650.
[16] Ami Hauptman and Moshe Sipper. 2005. *GP-endchess: Using genetic programming to evolve chess endgame players*. Springer.
[17] Phil Hellmuth. 2003. *Play poker like the pros*. Harper Paperbacks.
[18] Mirolsav Jelev and Stephan Koch. 2009. Genetic Blackjack. (2009).
[19] Daniel Kahneman. 2003. Maps of bounded rationality: Psychology for behavioral economics. *The American economic review* 93, 5 (2003), 1449–1475.
[20] Graham Kendall and Craig Smith. 2003. The evolution of blackjack strategies. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, Vol. 4. IEEE, 2474–2481.
[21] John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press.
[22] John R Koza, Forrest H Bennett, and Oscar Stiffelman. 1999. Genetic programming as a Darwinian invention machine. In *European Conference on Genetic Programming*. Springer, 93–108.
[23] William B Langdon and R Poll. 2005. Evolutionary solo pong players. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, Vol. 3. IEEE, 2621–2628.
[24] Frank Lantz, Aaron Isaksen, Alexander Jaffe, Andy Nealen, and Julian Togelius. 2017. Depth in Strategic Games. (2017).
[25] Andrés Pérez-Uribe and Eduardo Sanchez. 1998. Blackjack as a test bed for learning strategies in neural networks. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, Vol. 3. IEEE, 2022–2027.
[26] Ronald L Rivest. 1987. Learning decision lists. *Machine learning* 2, 3 (1987), 229–246.
[27] Jonathan Schaeffer, Joseph Culberson, Norman Treloar, Brent Knight, Paul Lu, and Duane Szafron. 1992. A world championship caliber checkers program. *Artificial Intelligence* 53, 2 (1992), 273–289.
[28] Yehonatan Shichel, Eran Ziserman, and Moshe Sipper. 2005. GP-robocode: Using genetic programming to evolve robocode players. In *8th European Conference on Genetic Programming*, Vol. 3447. 143–154.
[29] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and others. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.
[30] Herbert A Simon. 1972. Theories of bounded rationality. *Decision and organization* 1, 1 (1972), 161–176.
[31] David Sklansky. 1999. *The theory of poker*. Two Plus Two Publishing LLC.
[32] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. 2015. Solving Heads-Up Limit Texas Hold'em. In *IJCAI*. 645–652.
[33] Toru Tanigawa and Qiangfu Zhao. 2000. A study on efficient generation of decision trees using genetic programming. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 1047–1052.
[34] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jürgen Schmidhuber. 2009. Super mario evolution. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 156–161.
[35] Edward PK Tsang, Jin Li, and James M Butler. 1998. EDDIE beats the bookies. *Softw., Pract. Exper.* 28, 10 (1998), 1033–1043.
[36] Barkley Vowk, Alexander Sasha Wait, and Christian Schmidt. 2004. An evolutionary approach generates human competitive corewar programs. In *Workshop and Tutorial Proceedings Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife XI)*. Citeseer, 33–36.