Smart Ranger: A MCTS-inspired Approach for Optimal Strategy Approximation

Eric Tang
Statistics and CS
Harvard University
etang@college

Gerson Personnat
Statistics and CS
Harvard University
gersonpersonnat@college

Jessica Li
Statistics and CS
Harvard University
jessica_li@college

1 Introduction

Ranger is a single-player card game played with a standard shuffled deck of 52 cards. The player is tasked with selecting a subset of cards with size at most 20 from the deck, such as “Give me any heart, any king, or any ace." The deck is then dealt sequentially from the top of the deck until a card is dealt within that subset. All previous cards are given to the dealer. At this stage, the player is prompted for another subset, after which the same process repeats until the player has three cards in their hand. Following this, the dealer draws until they have at least five cards, and both the player and dealer form the best possible three-card poker hand. The winner is determined based on teen patti hand rankings, with the dealer winning ties and automatically winning with any three of a kind when the player specifically has three Aces.

The crux of the game’s complexity relies on its use of balance, creating a unique decision-making environment where the player must weigh the potential positive and negative outcomes of a given subset. The additional constraints of the game are designed to prevent trivial edge case strategies: the dealer drawing to at least five cards ensuring the dealer has a baseline hand and the dealer winning ties and often beating three Aces to prevent the player from directly going for the best possible hand in the game (3 A’s).

Our code can be found **here** (in order to access the link, please download the pdf).

2 Problem and Algorithmic Motivation

Ranger is especially compelling due to its large state and action space. With a shuffled deck of 52 cards, there are a vast number of possible card combinations, and the player’s hand evolves with each card drawn. To address these challenges, we aim to explore the problem using an algorithm inspired by Monte Carlo Tree Search (MCTS), whose primary goal is to build a search tree that reflects the most promising actions at any given state. Implementing pure MCTS is a computational intractable, as the sheer number of permutations of reasonable state-action pairs for each round makes simulating individual trajectories a herculean task.

To address these issues, we explore several modifications to the basic pure MCTS algorithm through domain knowledge-driven trimming of a reasonable action space conditional on a given state of the hands, pruning the action space without losing critical information. Although we will discuss our pruning algorithm in more depth in the following sections, the following rough principles will be the major themes used to guide our pruning:

1. **Symmetry:** During a runout of a game, we may take advantage of clear instances of symmetry. Consider the start of the game, for instance. From the player’s perspective, the suits are entirely symmetric–as there is no differentiation between clubs and hearts in the

hand rankings, the player similarly should have no reason to consider “all Hearts" as distinct from “all Clubs".

2. **Range Balance:** Since the player must use all of their given cards, a player’s specified range must achieve a balance between two goals: improving the player’s hand and blocking the dealer’s potential. For example suppose the player is presented with the following state: A Hearts, 2 Hearts (player hand), 4 Diamonds, 5 Spades (dealer hand).

   Here, any non-heart King is practically inconsequential to both the player and the dealer. There is practically no reason for the player to ask for such cards in such a state, as they run the risk of ending in a state such where their hand is A Hearts, 2 Hearts, K Spades, and having only a high-card hand instead of a flush, straight, or straight flush that they could have obtained by asking for some combination of Hearts and Threes. Thus, asking for inconsequential cards runs the risk of self-sabotage while not materially affecting the dealer’s hand, and we can reasonably exclude such cards from consideration.

3 Algorithmic Implementation

As noted above, our implementation of MCTS includes significant deviations from the pure version of the algorithm. Although our initial range selection relies on an algorithm reminiscent of the basic trimmed MCTS algorithm, subsequent rounds include more heuristic-based pruning following the two action space pruning principles outlined above.

3.1 Round 1

In the first round, the game begins with a full, shuffled deck with the player and dealer both having the same state of no cards. Thus, the optimal solution for this first step will be identical across all potential trajectories of the game. For the first round, then, we simulate win percentages for reasonable specified ranges to determine an approximately optimal solution, which we will hard-code for our investigations into the second and third rounds.

With the symmetric uniformity of the first round, we should only need to consider subsets of the 13 ranks as potential candidates for our range. In other words, we posit we do not have to consider suits since nothing differentiates a suit from another when the player has no cards.

Thus, instead of performing iterations over the action space of size

$$\sum_{i=1}^{20} \binom{52}{i} \approx 2.8 \cdot 10^{14},$$

we only need to consider subsets of at most 5 ranks of the 13 ranks (since each rank contains 4 possible cards), or an action space of size

$$\sum_{i=1}^{5} \binom{13}{i} = 2379,$$

a still-large, but more manageable action space to iterate over.

Due to the remarkably high variance of the game–the same action can lead to a breathtakingly large number of possible states–we need to run each node with a suitably large number of iterations to obtain any reasonable inference on the win probabilities for any given action. Thus, in order to maintain our search to within our computational bounds, we use a simple greedy heuristic after the first round–seeking to search for high straights, triples, or flushes–to identify the win probabilities for each action. Doing so, we obtain the optimal starting range as $(10, J, Q, K, A)$, simply the top five ranks, which checks out with our intuition.

3.2 Round 2: Dynamic Strategy with MCTS

Going into the second round, we no longer have the consistent state space in the first round where the player and dealer always have the same hands throughout every runout of the game, losing some of our simplifying notions of symmetry. This round is also the most intuitively complex–at this time, the player has only one card, offering them flexibility as to which hand to shoot for, making the optimal balance between obtaining a strong hand and blocking the dealer's possible hands remarkably opaque.

With these difficulty considerations in mind, the following principles were the most emphasized in our implementation to either serve as approximations that shrunk our search space or comments on particular hyperparameter selections:

1. **Action Space Pruning:** Once again, we have a monumental action space for our potential ranges, though we no longer can rely on symmetry arguments as we have done in the first round. For this round, we look towards our second algorithmic principle of range balance–noting a fundamental assumption that the only reasonable cards a player would ask for would serve either the purpose of improving the player's hand or hindering the dealer's possible hands. Thus, in our implementation, we only consider range combinations of helpful cards (defined as cards that allow the player to go for a flush, straight, or triple in the third round) and blocking cards (defined as cards that, if the dealer were to receive them, would give them a flush, straight, or triple).
2. **Balancing Player vs. Dealer Strength:** Including dealer-blocking cards adds an additional risk since blocking the dealer can act as a means of sabotaging the player's own hand strength. Thus, although there are very reasonable states in which a player would want to block the dealer (e.g. states in which the player is already ahead of the dealer and the dealer is one away from a hand that would beat the player's best hand), it would be silly for the player to construct their range solely focused on blocking the dealer and ignoring cards that helped them. Thus, for this round, we only consider subsets where at least half of the specified subset includes cards that improve the player's hand (some of these cards may coincidentally be dealer-blocking cards) with the remainder being the other dealer-blocking cards that do not help the player meaningfully.
3. **Simulation-Driven Optimization:** Similar to the first round, despite our ability to prune the action space down from just shy of the quadrillions into the thousands, the high variance and wide outcome distributions from the same action necessitate a large number of simulations for each action/node, slowing down the execution of our algorithm considerably. Even implementing parallelization to take advantage of the inherent iterative nature of simulating runouts, the sheer number of simulations required for reliable estimates of win probabilities is a remarkably difficult challenge and computational bottleneck.

The results from the second round are quite illuminating, as they follow alongside our intuitive explanations well. For instance, when there aren't many concerns of the dealer's hand, the algorithm presents a greedy-like range, typically including cards involved in obtaining a straight (which, in three-card poker, is rated higher than a flush). For instance, presented with the states

$$P : 10 \text{ Spades}$$
$$D : \emptyset,$$

the player has no discernable threat from the dealer. Our algorithm deigns to output the range $(9, 10, J, Q)$, notably ignoring the 8 rank despite it being a part of a potential straight with the player's 10. Although this potentially could be explained with some noise in our simulations, it's also likely explained through a domain knowledge-based explanation of the current state. On one hand, the 8 is the weakest card of the five potential cards involved in straights with the 10–not only is it involved in the lowest possible straight, it also restricts optionality in the third round, as the algorithm is essentially forced to only specify a 9 in order to make a hand (unless the 8 you draw happens to be the same suit, after which you can go for a flush), which is a remarkably narrow range that can allow the dealer to draw a large amount of cards and beat the player's straight. Although the 9 is also involved in the lowest possible straight, the algorithm can go for either an 8 or a J for the third round, giving the dealer fewer cards in expectation. Similarly, although obtaining a 10 would similarly restrict the player in the third round, the resultant hand (three of a kind) is much stronger than the weak straight, making the risk of giving the dealer more cards seemingly worthwhile. As a separate point, since the dealer is guaranteed to draw to at least five cards and has currently not drawn any, the player is allowed to be slightly more selective in the range they provide, as the dealer will draw cards anyways. Although this second note is difficult to precisely quantify due to the highly volatile nature of the game runouts, it acts as a qualitative heuristic that reasonably explains the current state situation.

With these concepts in mind, it seems to suggest that our algorithm is indeed deviating from basic greedy strategies simply going for the flush, straight, or triple and incorporating the state information from the dealer's hand as well, a promising sign for improved performance.

However, to further illustrate that the algorithm indeed incorporates the dealer's space in a nontrivial manner, we also consider the following runout where the dealer has a threatening potential hand:

$$P : \text{10 Spades}$$
$$D : \text{7 Diamonds, 7 Clubs, 4 Hearts,}$$

where our algorithm once again presents a deviation from the simple deterministic greedy strategy in returning the range $(7, 9, 10, J, Q)$. Indeed, once again, we see our algorithm deign not to include the pure greedy 8 and instead include the blocking card of 7–a card that does not help the player meaningfully but is still too important to ignore for the purpose of blocking the dealer. With the player hand being equivalent to the previous runout example, we demonstrate that our algorithm exhibits the balance between aggressive greedy and defensive blocking dependent on both the player and the dealer's hand–a nuanced strategy that showcases our algorithmic design principles that should compare well to the baseline deterministic strategies.

3.3 Round 3: Greedy Pruning Strategy

In the third round, the space becomes quite a bit simpler, as the benefit of blocking the dealer becomes much less apparent. Whereas blocking the dealer in the second round offered the player optionality in the third round to salvage their hand, the third round concludes the game–if the blocking card does not help the player, the player has purely just sabotaged their hand. Although there are certainly edge cases where this may not be the pure optimum, a more deterministic greedy-like strategy would be an appropriate approximation for the final round, allowing for dramatically improved computational efficiency for a relatively low performance loss.

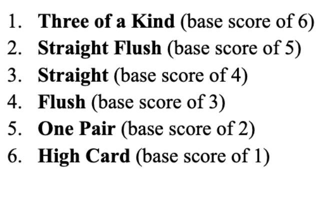
For instance, suppose the player is presented with the following space:

$$P : \text{10 Spades, 9 Clubs}$$
$$D : \text{7 Diamonds, 7 Clubs, 4 Hearts, 2 Spades.}$$

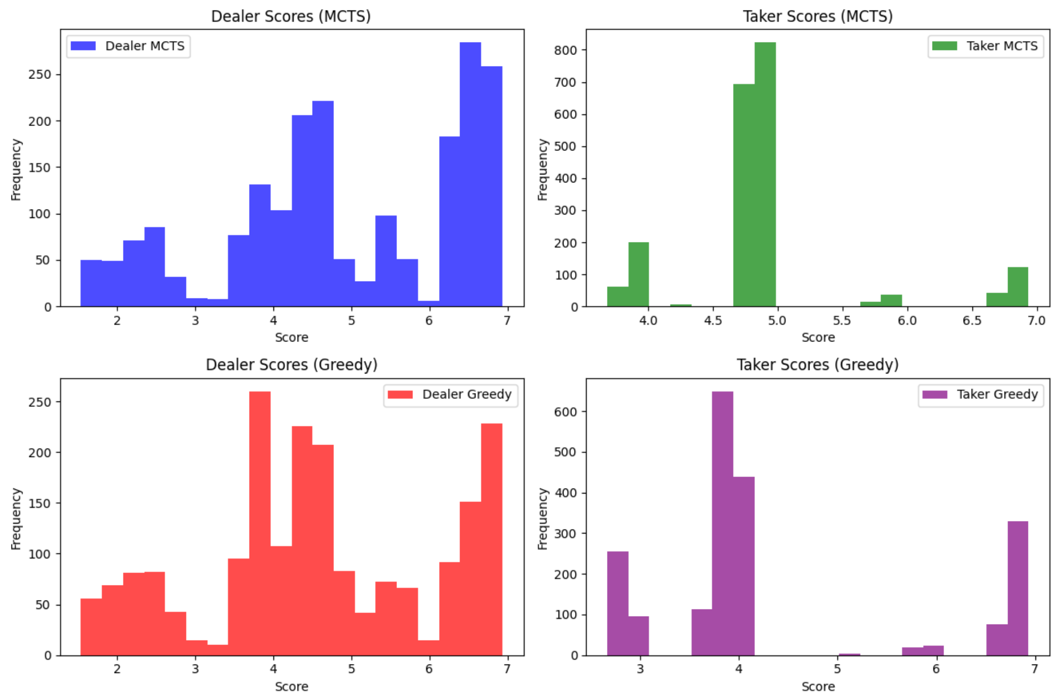
It is purely nonsensical for the player to block the dealer by specifying a 7 or 3, for instance, as the dealer would, at worst, have a pair of 7s and the player would just have a high card (note that blocking cards that also help the player are still considered). With this domain in mind, the optimal range for the large majority of runouts will simply approximate the greedy strategy for the player to obtain a high hand ranking.

4 Results

Using parallelization to speed up our simulation process, we ran 2000 simulations using our MCTS-inspired algorithm and a naive greedy strategy (a strategy that directly goes for triples, flushes, or straights, see driver code under "Baseline Strategy"), finding that the player wins about **50.5%** of the time using our modified MCTS algorithm versus only **39.15%** of the time using a greedy strategy, a dramatic improvement in win percentage.


Scoring System

1. **Three of a Kind** (base score of 6)
2. **Straight Flush** (base score of 5)
3. **Straight** (base score of 4)
4. **Flush** (base score of 3)
5. **One Pair** (base score of 2)
6. **High Card** (base score of 1)



For an analysis of the simulations themselves, the dealer scores are higher and more clustered towards the upper range with our modified MCTS algorithm compared to the naive greedy strategy. Such a trend indicates that our modified MCTS algorithm is slightly more picky than the greedy strategy in its ranges–although this results in the dealer acquiring more cards and stronger hands, our algorithm still demonstrated an increase in the win percentage overall, indicating an improvement in the balance between player and dealer strength; although we allowed the dealer hands to become marginally stronger, the improvement in the player hands made the algorithm net beneficial to the player.

Indeed, looking at the player scores, the player tends to be more tightly clustered around decently strong hands–predominantly straights–with our amended MCTS algorithm compared to the greedy algorithm which more often obtains flushes or worse. Interestingly, the greedy algorithm also manages to obtain three-of-a-kind type hands, the strongest category in the ranking, more often than our modified algorithm, demonstrating that simply going for player hand strength alone while being ignorant of the dealer's strength is detrimental (as those times when the player got a triple, it seems many times the dealer got a better triple). Again, this plot demonstrates the importance of our starting design principles–it is not sufficient to only consider the player's hand strength in this game, but to also consider the interactions between player and dealer as they evolve through the rounds.

5 Extension: 5-Card Poker

The 5-card poker variant introduces additional complexity due to the larger state space and adjusted scoring priorities. This extension was designed to test the adaptability of the MCTS-based strategy and explore how the optimal subset selection shifts when more cards are involved and there is more granular scoring.

5.1 Modifications in Implementation

1. **Iterative Hand Refinement:** Later rounds use MCTS to dynamically adjust the strategy based on the evolving game state. The algorithm prioritizes subsets that enhance the probability of achieving high-value hands like flushes or straights while minimizing the dealer's opportunities.

2. **Pruning for Computational Feasibility:** Due to the larger state space, we used more aggressive pruning techniques. Subsets that neither contribute to building a strong hand nor block the dealer were excluded. This ensures that computational resources focus on the most promising actions.

   The first round is equivalent to the three card version, so similarly, we find that the optimal strategy (which is confirmed by running MCTS on the initial state in the game tree, when the dealer and taker have no cards) is to greedly the highest ranks 10, J, K, Q, A.

   For subsequent rounds, we ran our MCTS algorithm under two pruning strategies. The first strategy restricted the set of possible actions at a node by taking a selection the remaining cards (not in the dealer or player's hand) and returning $\min(20, \text{number of selected remaining cards})$ subsets of the remaining cards. The set of remaining cards was restricted to be the set of "blocking" cards, which we defined to be cards where the suit or rank of that card is represented at least once in the dealer's hand and "helping" cards, where the suit or rank is represented at least once in the player's hand. This assumes, based on domain knowledge, that is preferrable for the player to request a larger subset of cards.

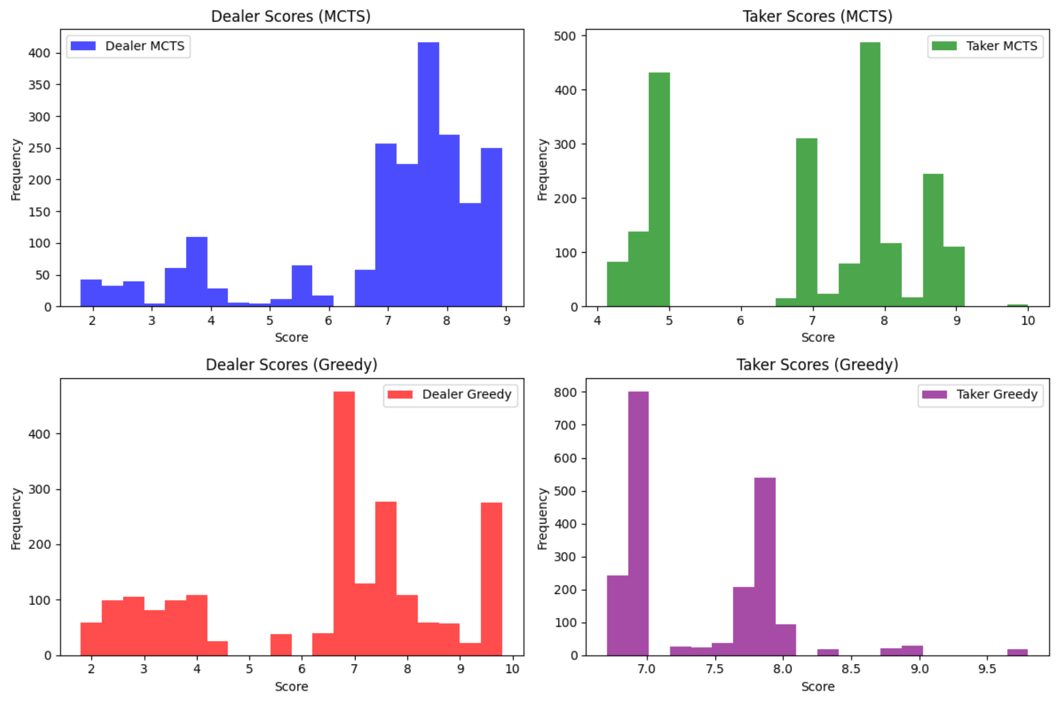
   The second pruning strategy aims to leverage the symmetry and suits and ranks. At each iteration, the MCTS explores a subtree of the game that we hypothesized to be optimal based on the initial results from the first strategy. At each round, the MCTS restricts the action space to consider all possible suit-rank combinations in the current dealer hand. For example suppose the player has a 10 of Spades after the first round. Then the MCTS algorithm will choose between actions requesting $(S, 10)$ (requesting any spade and a 10), $(10,)$ (requesting a card of rank 10) and $(S,)$, requesting any spade. Likewise, say the player has a Q Spades and 10 Spades after the second round. The player now has suits S and ranks 10 and Q represented in their hand. Then the MCTS algorithm will select between actions $(S,)(S, 10), (Q,), (Q, 10), (Q, S), (10,)$.

3. **Score Evaluation:** The 'convert_hand_to_value' function was adapted to evaluate 5-card hands with greater granularity. Fractional components, such as tie-breaking contributions from secondary kickers, were particularly important in distinguishing between hands of similar ranks. From a given node in the search tree, for a given state s, the MCTS algorithm chooses an action a to explore using the follower UCB (upper-confidence bound) score. The preceding state is chosen based on a random simulation of one round of the game. For example, if the dealer hand is a 10 Jacks. Then based for a given action (i.e. specifying a set of cards to request), we expand the current search tree to a child node by randomly simulating a round of the Ranger game. In this case, if the action is $(10,)$ (request a 10), then according to a random simulation, the preceding state might be the player hand having 10 Jacks and 10 of hearts with some dealer hand. We have a child node representing the state. To evaluate a node, we used a standard UCB bound:

   $\text{UCB}(s, a) = \frac{\text{wins}_{\text{child}}}{\text{visits}_{\text{child}}} + C \cdot \sqrt{\frac{\log(\text{visits}_{\text{state}})}{\text{visits}_{\text{child}}}}$, where $\text{wins}_{\text{child}}$ is the number of times the player has won from the child state, $\text{visits}_{\text{child}}$ is the number of times the algorithm has explored the child, and $\text{visits}_{\text{state}}$ is the number of times the algorithm has explored the current state.

   This UCB policy is also used for the third-card version. However, for the five-card version instead of using the number of times a player won to evaluate a node, we instead use the number of times a player has reached a score $> \alpha$ from some node for computational reasons. In later rounds of the game, the dealer may have many cards. To determine whether the player won or not entails computing the best 5-card hand that the dealer can get which requires enumerating over $\binom{n}{5}$ subsets if the dealer has n cards. When $n > 40$, $\binom{n}{5}$ is a practically large number, and in constrast to the three card games, $\binom{n}{3} << \binom{n}{5}$. In our final implementation, we set $\alpha = 7.5$ by observing the performance of different values of α over a few simulations.

5.2 Results



In implementation, our algorithm sometimes runs into odd local optima (e.g. settling on a singular non-relevant card, likely due to the noise from the simulations being too high–we could solve this theoretically with more iterations, but that would require more computational resources to be feasible). With a slight amount of human intervention (simply asking it to generate a more reasonable range), we obtain the following 2000 simulations. With the rerunning of the incorrect ranges outlined above, the algorithm obtains a win percentage of about **48.28%**, which, in comparison to our implementation of the greedy algorithm, is a little bit worse (our greedy algorithm achieves a win percentage of **59.54%**), indicating that the algorithm still has yet to handle the larger state space well enough for us to reach the greedy algorithm's performance. However, we do see traces of similar results from our analysis of the simpler three-card version of the game. Our modified MCTS algorithm is generally more picky than the greedy algorithm (although we get some iterations where the algorithm gets a weak hand like 4-5, likely due to the identification of the incorrect optima as explained above), the bulk of the iterations actually obtain more four-of-a-kind type hands in comparison to the greedy algorithm, perhaps indicating a slightly higher degree of the MCTS-based algorithm being slightly pickier than the general naive strategy when working as intended (this is also supported by the dealer scores in the MCTS-based algorithm being more biased upward than in the naive greedy case).

Scoring System

1. **Royal Flush** (base score of 10)
2. **Straight Flush** (base score of 9)
3. **Four of a Kind** (base score of 8)
4. **Full House** (base score of 7)
5. **Flush** (base score of 6)
6. **Straight** (base score of 5)
7. **Three of a Kind** (base score of 4)
8. **Two Pair** (base score of 3)
9. **One Pair** (base score of 2)
10. **High Card** (base score of 1)

6 Extension: 2-Player Poker

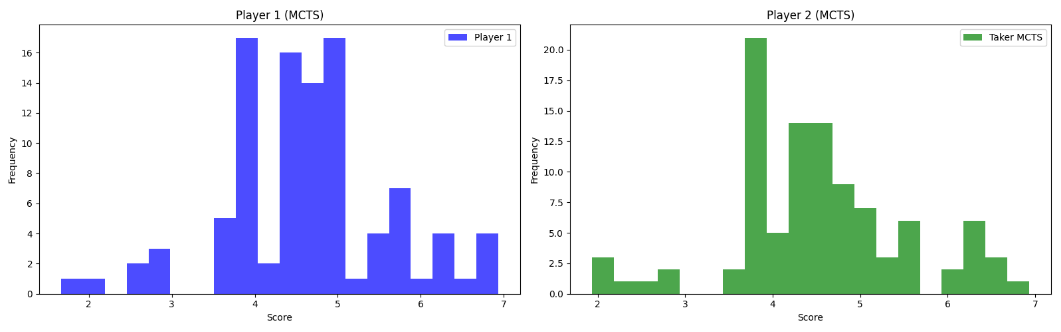
The two-player extension of Ranger introduces direct competition and new rules that require dynamic decision-making. To address balance issues, increase strategic depth, and ensure that players don't end up with large hands, we adapted how cards are allocated. If a card is drawn and matches the active player's specified subset, that card is given to the player, and the turn ends. If no match occurs, up to a maximum of three cards are drawn and given to the opponent. This rule also removes the guarantee that players receive a fixed number of cards per round, requiring a more dynamic and state-dependent strategy.

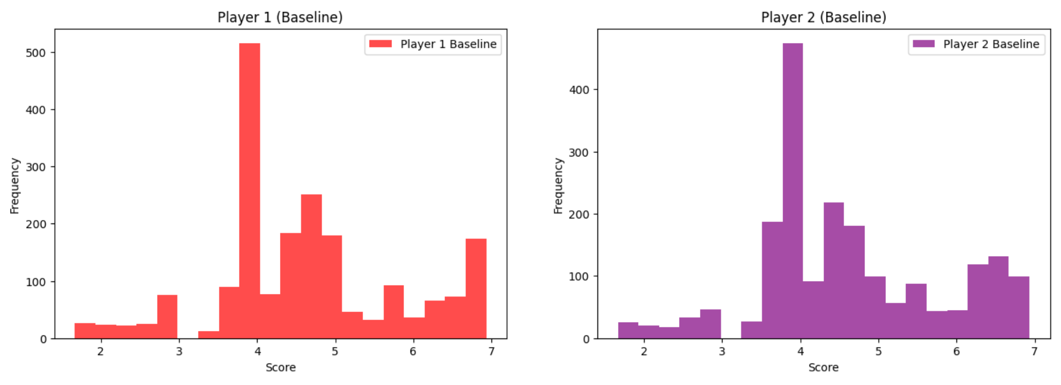
6.1 Modifications in Implementation

1. **State-Focused Subset Recommendations:** Given the uncertainty in hand sizes as well as the possibility of Player 2 having cards before their first turn, the algorithm for the 2-player game dynamically adapts to the current state of the game. Instead of hard coding strategies solely based on the round number, our algorithm dynamically adapts its recommendations based on the player's current hand and the opponent's potential hand.
   - Early and Mid Rounds: In early to mid rounds, if a player's hand is empty, the algorithm recommends a broad subset consisting of high-value ranks (10 $\rightarrow$ A) to maximize potential outcomes. Otherwise, the algorithm balances improving the player's hand and blocking the opponent's potential combinations. Subsets are dynamically pruned to 1) include cards that help the player's hand (e.g., completing a flush or straight) and 2) include cards that block the opponent's potential combinations (e.g., preventing a straight or triple). However, blocking is considered secondary to helping the player's hand and is included only when it aligns with helping the player.
   - Final Rounds: In the last round, the algorithm exclusively targets cards that improve the player's hand. Blocking the opponent is deprioritized, as it would limit the player's ability to maximize their final hand strength. This approach ensures the player focuses on achieving the strongest possible hand rather than exploring options that no longer provide value.

6.2 Results

To test the effectiveness of our MCTS algorithm, we compared 100 simulations of our MCTS algorithm (limited by computational constraints) to 2000 simulations of a baseline greedy algorithm.





With our MCTS-based algorithm, Player 1 won **60%** of the time (60 wins), Player 2 won **39%** of the time (39 wins), and there were ties 1% of the time (1 time). With the baseline greedy algorithm, Player 1 won **54.35%** of the time (1087 wins), Player 2 won **45.2%** of the time (904 wins), and there were ties 0.45% of the time (9 times).

The baseline strategy produces tightly clustered scores (4-5) which makes sense since it is deterministic in nature (both players aim for straightforward hands like three of a kinds or flushes). In contrast, the MCTS-based approach shows a wider spread of scores, particularly for Player 1, indicating dynamic adaptability and a balance between exploration and exploitation. This allows Player 1 to leverage the first-move advantage more effectively, with higher frequencies of scores above 5 compared to the baseline and a increased win percentage from 54.35% to 60%. Player 2's performance under MCTS shows increased variance, with occasional high scores (6) but also a greater presence of lower scores (2-3). This suggests that the strategy incorporates blocking more effectively but sometimes sacrifices hand strength as a trade-off.

These preliminary results seem to indicate that the 2-player game is a game more focused on initiative as opposed to reactivity–Player 1 getting to specify ranges first seems to perform better than Player 2. In our continued work, we aim to study whether or not the second player can indeed deviate in their behavior to play in reaction to the first player, perhaps through a self play-type algorithm.

7 Next Steps and Future Work

Going forward, we want to conduct a more nuanced comparison of MCTS and naive strategies to understand the significance of the difference in win rates, and explore parameter fine-tuning to see how it could impact the win rates and computational efficiency. Additionally, our analysis of the 5-card variant is somewhat surface level, and we want to explore this and other variants (like multi-player game play) in more depth.

1. **Overarching Assumptions**: A fundamental assumption we made (our second design principle) is that a player would only consider cards helping their own hand or blocking the dealer, a semi-greedy-esque line of thinking that may have biased our algorithm to prefer ranges that coarsely emulate the greedy algorithm. Perhaps there is a way to think about the future where a card doesn't directly block the dealer nor help the player but could feasibly be included in the player's range in a given round.
2. **Three Card Base Game**: In addition to the pruning and symmetry arguments we used to guide our simulations, our overarching algorithm made a few simplifying assumptions for the sake of computational efficiency with our current resources. In particular, as the second round was already computationally straining, implementing a complete simulation of the third and final round as an additional layer would have been infeasible with our current timeline, prompting us to implement our more greedy-based approximation. Further study would involve a more complete investigation into the third round to ensure that our greedy-based approximation is indeed sound.
3. **Five Card Game**: Like in the base three card game of Ranger, we also assume that the player should consider cards that greedily help them acquire the best possible hand. We also assume that the player should use a strategy only considering the cards that most help its hand in the current round. Future work could consider how we can incorporate more forward-looking strategies in earlier rounds by thinking of ways to prune the action space in a less greedy manner. In addition, in our implementation, we essentially use the same pruning strategy across round, so we could consider what happens if we evolve the pruning of the strategy space across rounds.
4. **2-Player Game**: Future improvements could focus on smarter, opponent-aware strategies, such as refining pruning techniques, adding long-term planning, and exploring ways to balance the advantage of going first. Since Player 1 consistently outperforms Player 2 due to the initiative of specifying ranges first, future work could explore more adaptive strategies for Player 2, such as reactive blocking or self-play-type algorithms. Alternative rule changes, like giving Player 2 additional advantages, could also help improve fairness and balance.